MASTER EN SCIENCES INFORMATIQUES

Mesures de performances de UNIX sous une charge universitaire

Adans, Jean-Paul; Vercheval, Jean

Award date:
1979

Awarding institution:
Universite de Namur

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Mesures de performances de UNIX sous une charge universitaire

PROMOTEUR : M. NOIRHOMME

JEAN-PAUL ADANS
JEAN VERCHEVAL

MEMOIRE PRESENTE EN VUE DE L'OBTENTION DU DIPLOME DE LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1978-1979

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Mesures de performances de UNIX sous une charge universitaire

PROMOTEUR : M. NOIRHOMME

JEAN-PAUL ADANS
JEAN VERCHEVAL

MEMOIRE PRESENTE EN VUE DE L'OBTENTION DU DIPLOME DE LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1978-1979

PREFACE

Nous voudrions remercier Madame M. Noirhomme pour les directives qu'elle a bien voulu nous donner tout au long de notre travail.

Nous sommes très reconnaissants au Department of Computer Science du Queen Mary College, et en particulier à Messieurs J. Rowson et G. Colouris, pour nous avoir permis une première prise de contact avec UNIX.

Nous tenons à exprimer toute notre gratitude envers Monsieur J. Demarteau, qui nous a aidés lors de la mise en route de UNIX, et qui a manifesté beaucoup de bienveillance lors de nos nombreux problèmes de maintenance.

Nous remercions Monsieur E. Milgrom pour les conseils qu'il nous a prodigués.

Nous remercions également Messieurs J. Ramaekers et Ph. Van Bastelaer pour avoir mis à notre disposition les benchmarks élaborés par la Commission Plan Calcul des Facultés.

Nous sommes particulièrement reconnaissants à Monsieur Ph. De Rivet pour avoir bien voulu discuter avec nous certains aspects de la métrologie informatique propres à notre problème.

Enfin, nous tenons à remercier Monsieur le Docteur M. Noël qui nous a permis d'utiliser les ordinateurs de son laboratoire.

TABLE des MATIERES

Pages

Pages

Pages

INTRODUCTION

Nous avons voulu construire un outil de mesure de performances pour le futur système S3 des Facultés Universitaires de Namur. Plus encore, nous avons voulu montrer une combinaison possible des techniques de la métrologie informatique avec des méthodes d'analyse statistique.

Il va de soi que l'élaboration d'un tel outil est une tâche complexe, et il nous a été impossible d'envisager un travail comparable à celui réalisé par les équipes spécialisées de certains constructeurs.

Néanmoins, en limitant d'avance notre travail, un tel but nous a semblé réalisable.

Nous nous sommes donc orientés vers la réalisation d'un outil de mesure et d'analyse de ce que nous appellerons provisoirement le temps de réponse. Cet outil devait nous permettre de saisir la sensibilité du système par rapport à certains paramètres logiciels et, du reste, d'évaluer l'importance relative de ces paramètres.

Notre étude s'est déroulée en plusieurs étapes. Tout d'abord, il nous fallait acquérir des notions de métrologie et d'analyse statistique appropriées à notre problème.

Si, en ce qui concerne ces points, la littérature est abondante, elle se cantonne généralement dans des sujets spécialisés, en relatant l'expérience propre des auteurs dans des cas particuliers.

Nous avons donc dû en synthétiser des règles de comportement suffisamment générales pour nous permettre la réalisation de notre objectif.

Les deux premiers chapitres de cet ouvrage sont destinés à procurer au lecteur ces notions de base générales.

Parallèlement à cela, nous devions nous munir d'une bonne connaissance du système informatique S3. Il s'agit d'un ordinateur largement répandu dans les milieux

universitaires, en l'occurrence d'un PDP 11/45 de la firme Digital Equipment Corporation.

Le système d'exploitation est également un sujet d'étude et un outil pédagogique souvent rencontré dans les universités : il s'agit ici du système UNIX (1), réalisé dans sa version actuelle vers la fin des années '60 par Dennis M. RITCHIE et Ken D. THOMPSON, des Bell Telephone Laboratories. Ce système est exploité sous licence de la Western Electric Company.

Après un stage qui nous a permis une première prise de contact avec UNIX, nous avons été confrontés à de nombreux problèmes, tant matériels que logiciels, lors de la mise en route et de la maintenance de ce système. Cette situation nous a conduits à constamment approfondir nos connaissances, ce qui, tout en exigeant une grande partie de notre temps, n'a pas été directement productif en ce qui concerne notre travail.

Le chapitre trois du présent ouvrage est consacré à ce système informatique. Il veut donner au lecteur intéressé une explication des mécanismes mis en oeuvre.

Du reste, ces explications servent de toile de fond aux chapitres suivants, qui ne manquent pas de faire appel à ces notions.

Ensuite, nous avons jugé utile de traiter en détail de la charge du système. Celle-ci est indispensable pour l'exécution de mesures de performances. La difficulté a résidé dans le fait que la charge réelle du système informatique S3 n'est pas encore connue, S3 n'étant pas encore installé.

---

(1) UNIX est une marque déposée par les Bell Laboratories.

Les résultats que nous avons obtenus sont donc valables pour la charge que nous avons construite. Ils seront sans doute à revoir une fois la charge réelle connue.

Enfin, nous exposerons l'outil que nous avons réalisé et montreront à l'aide d'un exemple, comment il peut s'utiliser.

Chapitre I

LES MESURES DE PERFORMANCES

Jean-Paul ADANS

Définir la métrologie des systèmes informatiques n'est pas une chose facile. Le champ d'application de cette technique est très vaste et comprend différents secteurs.

Délimiter les mesures de performances que l'on veut mettre en oeuvre demande plusieurs préliminaires :

- Définir l'environnement des mesures.

Il est en effet primordial de connaître ce que l'on veut mesurer. Cette définition doit pouvoir cerner ce qu'il est possible de capter : les mesures doivent s'appliquer à quelque chose, et qui dit mesure dit lecture, collecte d'une grandeur appréciable.

- Définir le but des mesures.

Pour pouvoir élaborer une méthode de mesure, il faut clairement cerner le but de ces mesures. Elles peuvent servir plusieurs objectifs et sans but précis, elles ne procurent qu'une accumulation inutile de données.

- Définir les possibilités.

La métrologie des systèmes informatiques est souvent onéreuse. Soit qu'elle demande l'interruption du travail normal d'un centre de calcul, soit qu'elle nécessite l'élaboration de logiciels complexes ou l'acquisition de matériels très spécialisés.

I.1 Objectif des mesures de performances.

Lorsqu'on projette de réaliser des mesures de performances, c'est que l'on a été amené à se poser certaines questions. Afin de pouvoir y répondre par la métrologie, il faut que ces questions soient clairement exprimées et que les éléments du système qui deviennent intéressants pour cet objectif soient bien définis.

L'objectif permettra de réduire la palette des mesures possibles à priori, et, de manière conséquente, l'outillage qui en deviendra nécessaire.

Une fois défini cet objectif, il faut pouvoir exprimer le cadre dans lequel on se place. Ceci se fera grâce à un modèle.

Le modèle reflète la compréhension que l'on a de la réalité, c'est-à-dire qu'il tente d'exprimer les liens que l'on suppose exister entre ce que l'on veut étudier et certaines contraintes significatives pour ce comportement du système.

I.1.1 Modèle métrologique.

Le modèle, expression formalisée des liens de cause à effet, comprend plusieurs éléments :

- variables de sorties, qui sont les variables que l'on veut mesurer et qui se présentent comme effet;
- variables d'entrées, c'est-à-dire des grandeurs que l'on présume être à l'origine d'éventuelles variations des variables de sortie.

De cela, il ressort que ces variables doivent être ou doivent pouvoir se représenter par des grandeurs significatives. De plus, si on veut étudier la variation des variables de sortie en fonction des paramètres d'entrée, ceux-ci doivent être influençables directement, leur valeur doit être connue et stable.

A ce modèle peuvent encore s'adjoindre des variables structurelles, i.e. des grandeurs que l'on estime importantes pour l'explication des phénomènes que l'on étudie, sur lesquelles cependant on ne désire pas agir. Elles sont là en quelque sorte pour rappeler leur existence.

Considérons par exemple une série de mesures que l'on envisage d'appliquer au taux d'activité de l'unité de transfert. Ce taux figurera comme variable de sortie du modèle. Afin de déterminer les variables d'entrée, il faut connaître les facteurs qui peuvent influencer la variable de sortie : taille et distribution sur support externe des fichiers, organisation de la mémoire paginée, etc...

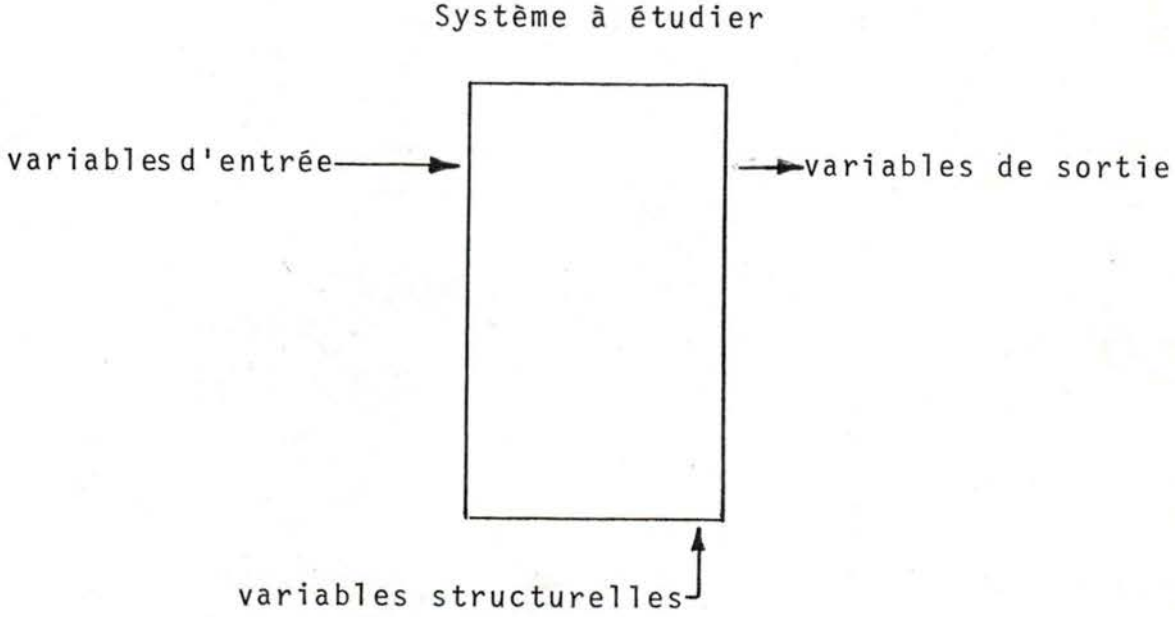


Fig. II.1 Schématisation du modèle

I.1.2 Buts de la métrologie.

La métrologie des systèmes informatiques peut être mise en oeuvre dans plusieurs buts : amélioration d'une exploitation ou d'une application existante, définition des options de base d'une nouvelle application, ou élaboration des critères de choix pour une nouvelle installation.

Il va de soi que les moyens à envisager sont très différents selon le type de situation dans laquelle on se trouve. L'objectif détermine donc les limites de la métrologie.

I.2 L'objet des mesures.

Lorsque la question à laquelle la métrologie doit répondre est formulée, il faut trouver ce que l'on va effectivement mesurer.

Dans certains cas, ce problème est rapidement résolu, en particulier lorsque la question est très précise.

Dans d'autres cas cependant, si par exemple les désirs sont exprimés par le gérant du centre de calcul, l'ingénieur système chargé des mesures doit lui-même déterminer les grandeurs qui sont significatives par rapport au problème posé.

I.2.1 Le système informatique comme ensemble de ressources.

Quand on considère le fonctionnement d'un système informatique, on voit qu'il procure certains services à ses utilisateurs. Ces services font appel à différents sous-ensembles d'éléments du système que l'on qualifie généralement de ressources.

Ces ressources sont soit matérielles (unité centrale, mémoires centrale ou auxiliaire, lignes de communication), soit logicielles (moniteur, éditeur, compilateur) et sont toujours des éléments indispensables à l'avancement des travaux. Comme de plus elles existent en nombre restreint dans le système, apparaissent des problèmes de concurrence lorsque plusieurs travaux demandent la même ressource.

Ce problème est absorbé par des dispositifs de partage de ressources.

Conséquemment à la concurrence, on voit apparaître le concept de consommation. Celle-ci s'exprime en différentes unités, suivant le type de ressource à laquelle elle se rapporte.

Pour l'unité centrale, on parlera de temps écoulé, tandis que pour la mémoire, on considérera le produit "espace occupé par temps d'occupation" (Bien que cette dernière ressource soit une des plus délicates à étudier dans un environnement à mémoire paginée ou segmentée).

Ce qui précède nous permet de considérer déjà un premier type d'entités mesurables : les unités de consommation de ressource. Ces unités sont très diversifiées et plusieurs tentatives pour ramener la consommation à une même unité apparaissent comme douteuses, et aucune standardisation n'est à l'heure actuelle possible en ce domaine.

I.2.2 Files d'attente et réseaux de files d'attente.

La concurrence des travaux présents dans le système (dans un environnement multiprogrammé), amène à un second type d'entités mesurables. En effet, elle nécessite des dispositifs de gestion qui seront le plus souvent des files d'attente.

La littérature est abondante sur ce sujet, et des modèles analytiques permettent l'étude de systèmes de

files d'attente. Un réseau de files d'attente pourra être représentatif du comportement des travaux à l'égard des différentes ressources du système.

I.2.3 Temps de réponse.

Lorsqu'on se place au niveau de l'utilisateur, on peut considérer le système dans son ensemble comme un poste de service. L'utilisateur ne s'intéresse qu'à ses propres travaux. Suivant le cas, il considérera soit le temps écoulé entre la remise de son paquet de cartes perforées et la réception du listing d'exécution, soit le temps écoulé entre la fin de la frappe d'une ligne de commande au clavier de son terminal et le début de l'impression du message d'exécution sur son écran ou sa télétype.

Ce temps sera appelé temps de réponse. Il s'agit ici de faire une distinction qui sera explicitée par la figure II.4.a. En effet, on peut se placer à différents niveaux pour mesurer des temps propres à cette situation.

Dans le chronogramme de la figure II.4, les t_i représentent :

t_1 : début de la frappe de la commande

t_2 : fin de la ligne de commande (éventuellement caractérisée par un caractère spécial, <CR>), et début de transmission du message.

t_3 : fin de la transmission et début du traitement

t_4 : fin du traitement et début transmission de la réponse

t_5 : fin de la transmission et début de l'impression de la réponse

t_6 : fin de l'impression du message.

Ce cas général vaut pour une application en temps partagé, mais peut facilement se transposer pour une application de type batch.

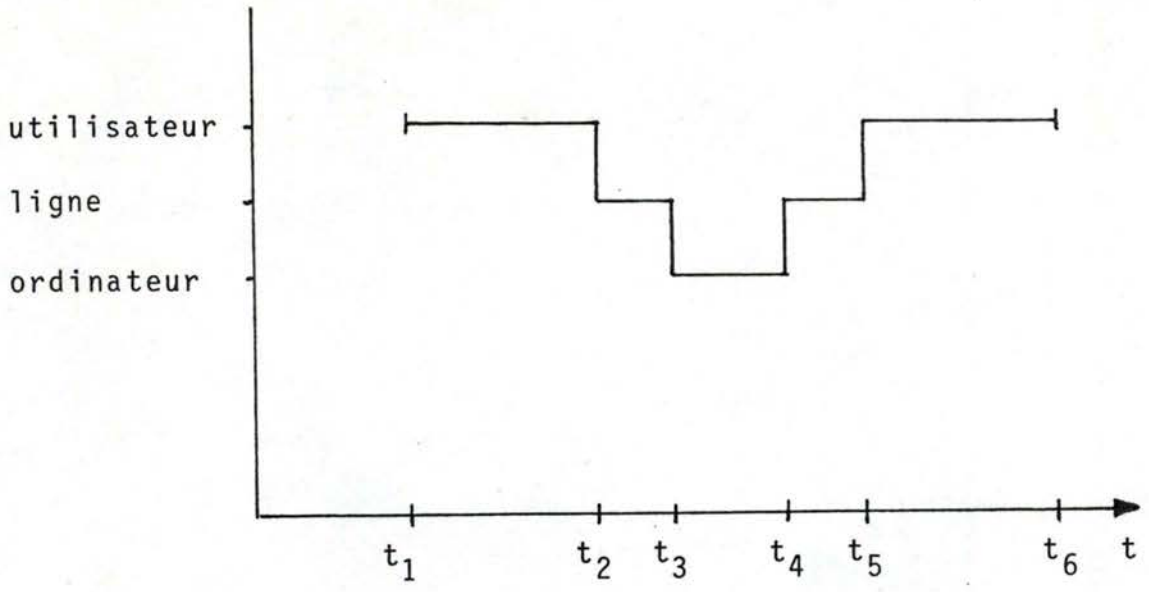


Fig. II.4.a Chronogramme d'un échange de message entre terminal et ordinateur

Suivant la situation, ce schéma est modifié par

- l'intelligence des terminaux :
  si le terminal ne dispose pas de tampons, les temps t_1 et t_2 coïncident, de même que t_4 et t_5;
- le type des lignes utilisées (half, full duplex);:
- le mode de transmission (synchrone, asynchrone);
- le type de messages envoyés :
  il est possible que le traitement continue pendant la transmission et l'impression du message de réponse; auquel cas il faut considérer des instants $t_4°$, $t_5°$, $t_6°$, pour l'envoi du premier message et t_4', t_5' et t_6' pour l'envoi du dernier message de réponse (fig. II. 4.b).

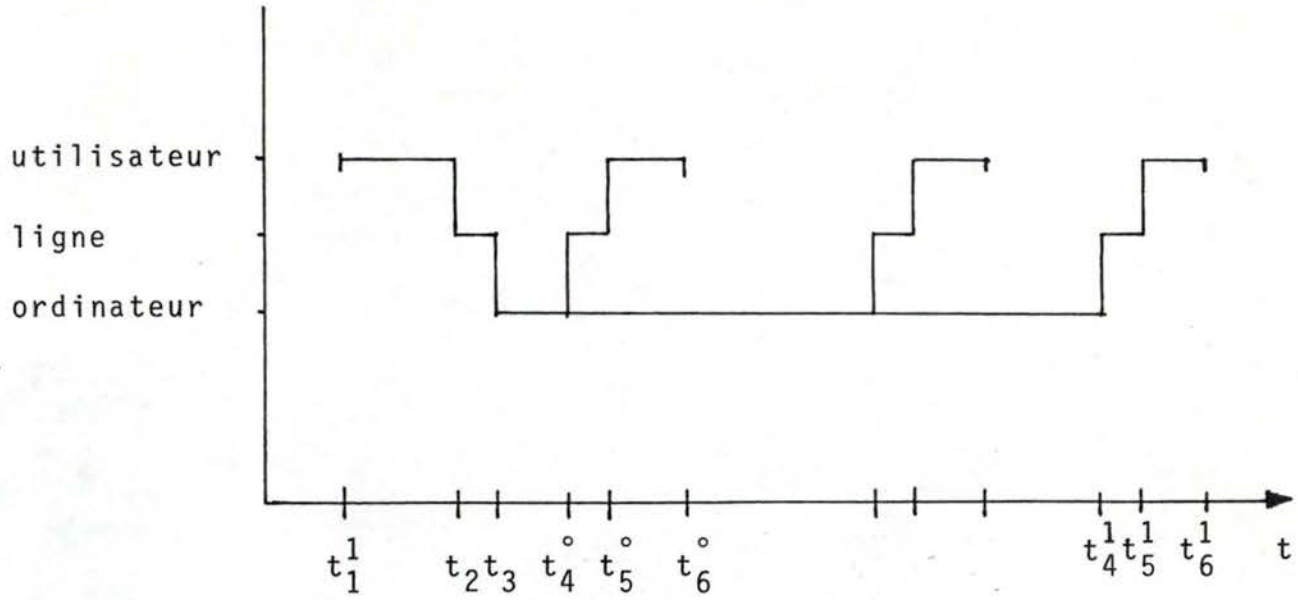


Fig. II.4.b Chronogramme modifié

Seule la connaissance précise d'une situation typique permet de décrire exactement ce phénomène.

On peut cependant dégager quelques constantes : la différence $t_4^\circ - t_2$ est généralement appelée temps de réponse alors que l'intervalle entre le début de la frappe d'un message par l'utilisateur et le début de la frappe du message suivant constitue le temps de cycle.

Ce temps de cycle est donc constitué par le temps écoulé entre deux instants t_1 et t_6^1 successifs augmenté d'un intervalle aléatoire, appelé temps de réflexion. Ce dernier représente le temps mis par l'utilisateur pour préparer son prochain message après réception du précédent.

On peut schématiser ce processus par un chronogramme tel que celui de la figure II.5.

Dans la figure II.5, l'intervalle $t_2^x - t_3^x$ représente un temps de réponse, tandis que l'intervalle $t_o^x - t_o^{x+1}$ donne un temps de cycle.

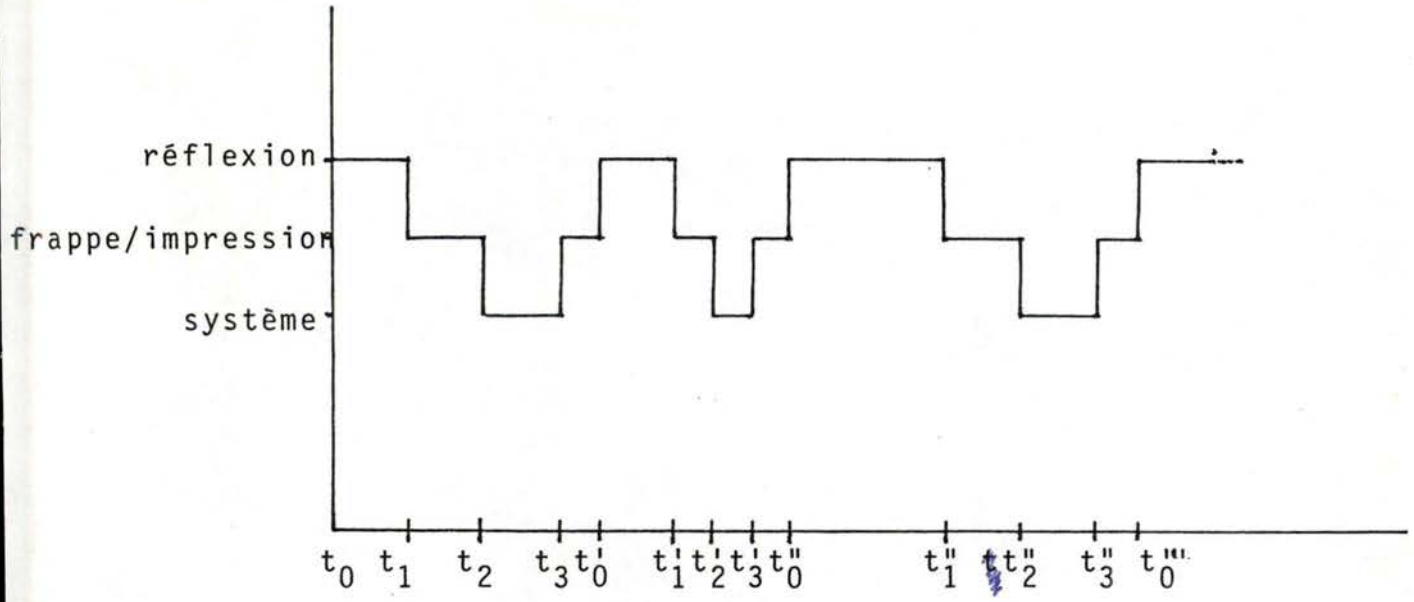


Fig. II.5 Chronogramme du point de vue de l'utilisateur

Du point de vue externe ce sont ces grandeurs qui sont intéressantes pour l'utilisateur. Le gérant du centre devra s'efforcer de garder ces temps dans des limites acceptables.

On considère parfois également la charge absorbable, i.e. pour un utilisateur au terminal, le nombre de travaux exécutables par unité de temps (ou le temps nécessaire à l'exécution d'un nombre fixe de travaux). Cependant, cette dernière grandeur n'est pas utile en temps partagé, en raison du caractère aléatoire du temps de réflexion. Elle est la performance typique d'un système fonctionnant en mode batch, i.e. ou les travaux sont introduits par lots au moyen de cartes perforées.

I.2.4 Choix de l'objet des mesures.

Dans les paragraphes précédents, nous avons énuméré ce que nous pensons être les différentes grandeurs mesurables dans un système informatique.

Sans avoir voulu être exhaustifs, nous avons parlé de ce que nous appellerons les mesures internes, c'est-à-dire celles qui peuvent fournir à l'ingénieur système des renseignements quant au fonctionnement du système.

Ce genre de mesures doit permettre d'agir sur certains facteurs que l'on peut conclure décisifs.

L'utilisateur banal ne s'intéresse qu'au déroulement de ces travaux. Les critères qui lui sont importants peuvent également fournir des renseignements utiles au gérant.

Le bon fonctionnement du centre de calcul et la satisfaction de ses utilisateurs sont des finalités que doit atteindre l'équipe chargée de sa gestion.

Les exigences des utilisateurs peuvent fournir une ligne de conduite permettant de déterminer les performances. Dans d'autres cas où l'utilisateur n'est pas directement impliqué (ex. : choix d'une nouvelle installation), c'est sur base de l'expérience antérieure et de connaissances théoriques qu'il faut trouver ce que l'on va mesurer.

Dans tous les cas, seule l'approche formalisée à l'aide du modèle métrologique permet de déterminer avec précision l'ensemble des grandeurs qui sont significatives pour l'objectif fixé.

I.3 Les méthodes de mesure.

Quand les variables à mesurer sont déterminées, il faut trouver comment en saisir la valeur et la variation.

Dans certains cas, la technique à utiliser devient rapidement évidente. Ce n'est pas toujours ainsi.

Remarquons dès à présent qu'il est parfaitement inutile de savoir qu'à tel moment, telle variable avait telle valeur. Il faut plutôt pouvoir reconstituer l'évolution de la variable dans le temps. De plus, il faut pouvoir

situer cette évolution dans un contexte, sans quoi l'interprétation deviendrait fastidieuse.

Il est possible toutefois de considérer certaines moyennes, comme par exemple le taux d'activité du processeur central ou des processeurs périphériques. Cette information procure cependant un intérêt limité.

On essayera dès lors de constituer des traces, i.e. une représentation de l'évolution dans le temps des entités mesurées.

Rappelons qu'il s'agit ici de considérations générales qui doivent être adaptées à chaque cas particulier.

I.3.1 Outils de mesures.

Ces outils doivent pouvoir capter la valeur de certaines variables.

D'une part, ces variables sont liées à des phénomènes logiques. En grande partie, ceux-ci concernent le système d'exploitation. Par exemple, si on veut connaître le nombre de défauts de page, il faut considérer les passages par le sous-programme de pagination.

Pour ce faire, il suffit d'inclure en un endroit bien choisi, une séquence d'instructions réalisant par exemple l'incrémentation d'un compteur.
De telles séquences d'instructions portent le nom de sondes logicielles et peuvent être incorporées au texte du système d'exploitation, soit au moment de la génération, soit par insertion dynamique. Le choix dépend ici des possibilités que l'on a.

L'ensemble de ces sondes fournit des informations qui doivent être traitées par programme. Le rythme de ce traitement dépend de la vitesse d'acquisition des données et de la taille des compteurs utilisés. Une fois de plus, le pouvoir de résolution de l'outil dépendra de l'objectif assigné.

Lorsque l'on combine les sondes et programmes de traitement avec des possibilités de mémorisation secondaires (bande magnétique), on obtient un moniteur logiciel.

Généralement, ces moniteurs ne réalisent qu'un pré-traitement des données avant mémorisation, l'analyse complète se faisant en un autre moment.

La figure II.6 donne le schéma de principe d'un moniteur logiciel.

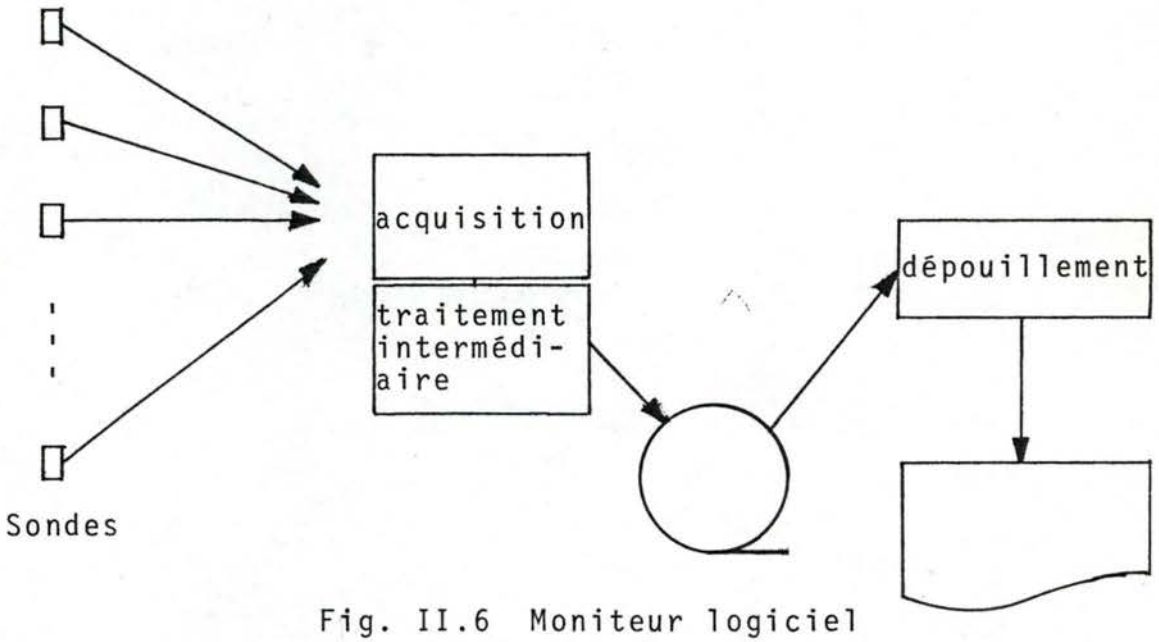


Fig. II.6 Moniteur logiciel

D'une structure semblable, les moniteurs matériels réalisent également une trace d'évolution, mais au départ de sondes captant des différences de potentiel du courant électrique.

Ces capteurs sont placés directement sur les circuits imprimés de l'ordinateur, le plus souvent sur le panneau de connexion arrière. L'interprétation des signaux reçus est d'autant plus difficile que ce dispositif se trouve à l'extérieur et ne connaît donc pas le contexte.

Idéalement, une interface d'échange réalise la synchronisation entre les phénomènes logiques au sein du système informatique et les événements physiques dans ses composantes.

On comprend aisément qu'en règle générale, ces moniteurs, matériels ou logiciels, sont très spécialisés et valables pour un nombre relativement restreint d'ordinateurs.

I.3.2 Techniques de mesures.

On distingue deux techniques de mesures. D'une part, la mesure sur événement ("event driven") enregistre la situation de certaines variables au moment de l'occurrence de certains événements (ex. : défaut de page).

D'autre part, on peut effectuer un ensemble de mesures à intervalles réguliers ("snap shot"), à l'aide d'une source de synchronisation.

De manière générale, le choix de la technique dépend de l'entité mesurée, ainsi que de l'objectif.

Il faut cependant veiller à certains points communs :

1) La collecte de données ne doit pas perturber, ou du moins le moins possible, le fonctionnement normal du système. Dans le cas d'une perturbation, celle-ci doit être quantifiable.
2) Le dispositif de mesure doit être fidèle : il faut que tous les éléments nécessaires soient captés et aucune information ne doit être perdue.
3) La distorsion doit être évitée au maximum : les vitesses différentes au niveau de l'acquisition et du traitement de la mémorisation des données peuvent engendrer des décalages dont il faut tenir compte.
4) L'accès aux variables à mesurer doit être garanti. La majorité de celles-ci sont en effet protégées à différents niveaux.

I.3.3 La charge.

Pour que l'on puisse mesurer un système informatique et pour que les résultats obtenus aient un sens, il faut que le système se trouve dans une situation telle qu'il reçoive des demandes, que ses ressources soient sollicitées.

L'ensemble des demandes aux ressources est appelée la charge.

La charge est représentative de ce qui se fait avec l'ordinateur dans un site donné. Cette charge différant suivant le type d'applications auxquelles le système est utilisé, les mesures différeront également par site.

La construction de la chagre constitue donc un élément très important, que nous développerons au cours du chapitre V.

I.4 Résumé des étapes.

Considérons ici uniquement les étapes de la mise en oeuvre de mesures de performances.

L'organigramme de la figure II.7 ci-après, schématise le procédé.

On voit que ce procédé doit être dynamique et permettre des retours en arrière : si le modèle ne peut être complété, il faut éventuellement approfondir la connaissance que l'on a du système ou reconsidérer la formulation des questions posées.

De même, si les mesures ne peuvent conduire à des résultats satisfaisants, il faut reprendre certaines étapes de la démarche.

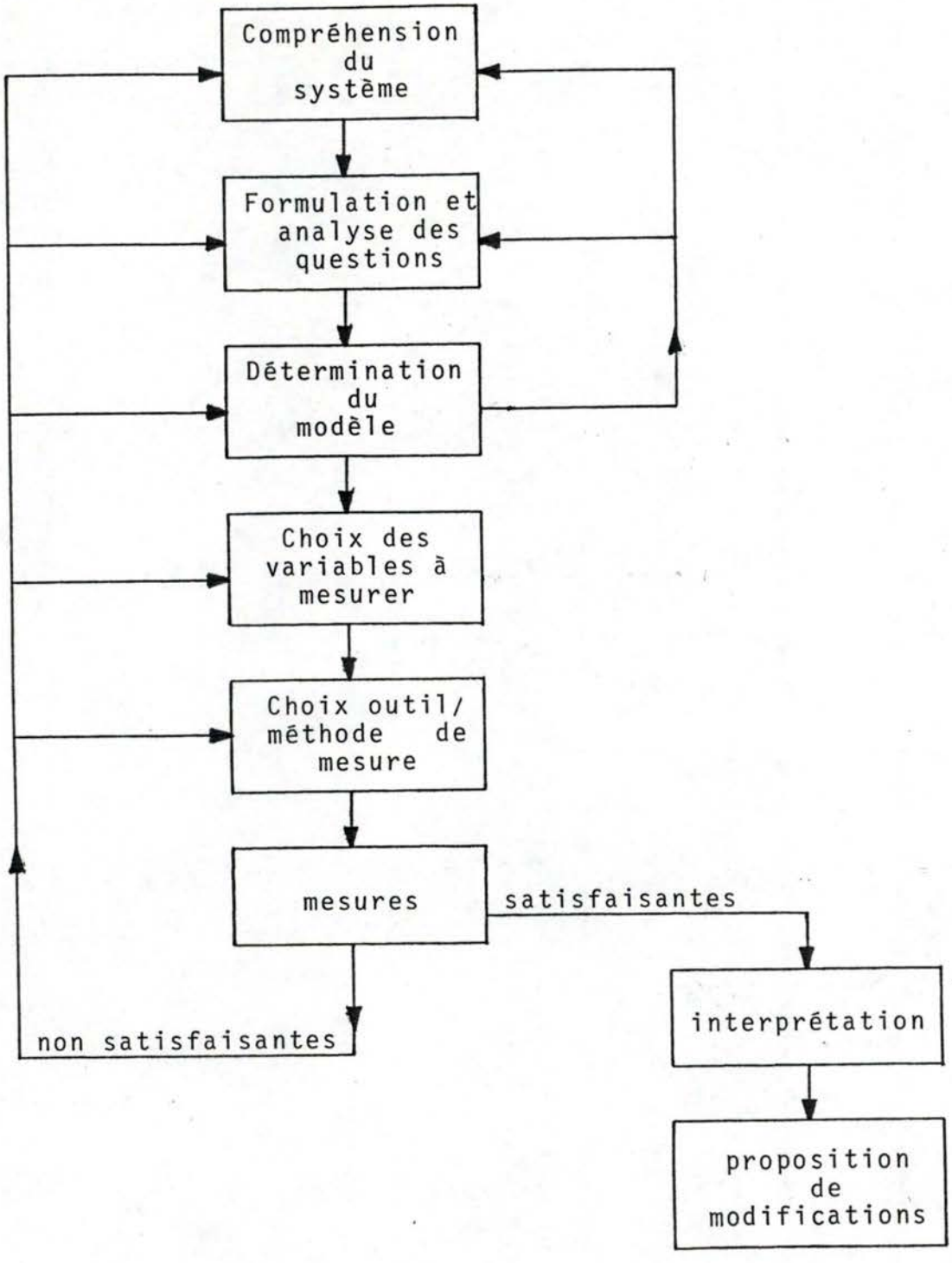


Fig. II.7 Etapes de la mise en oeuvre d'un outil métrologique

C'est consciemment que nous avons omis ici de mentionner la construction de la charge. Celle-ci se pose en quelque sorte en parallèle, car, pour des raisons qui apparaîtront plus tard, elle ne peut pas faire partie intégrante de cette démarche, même si elle peut intervenir implicitement dans les divers stades de définition du problème.

I.5 Définition des performances de S3.

Nous avons décrit ce que nous avons appelé les performances externes du système informatique, en nous plaçant dans la situation d'un utilisateur à son terminal. En fait, c'est ce type de performances qui nous intéresse.

Etant donné les facteurs humains qui entrent en ligne de compte (fautesde frappe, temps de réflexion, commandes erronées), il ne nous paraît pas utile de mesurer la quantité de charge réalisable par l'utilisateur.

Par contre, celui-ci sera fortement concerné par les variations de temps de réponse. Le gérant du centre de calcul s'efforcera en général de garantir la rapidité de réponse et ainsi la satisfaction de l'utilisateur (bien que cela soit très subjectif).

I.5.1 Définition des performances.

Parmi les différents temps qu'il est possible de mesurer, il nous faut choisir celui qui nous paraît le plus apte à répondre à la question que nous nous posons.

Comme critères de choix, il faut également tenir compte des possibilités. Parmi les différentes solutions, certaines sont inacceptables car elles nécessitent des matériels ou des logiciels trop complexes.

Ainsi, pour mesurer le temps de réponse proprement dit, il faudrait mettre en oeuvre soit une liaison avec des terminaux suffisamment intelligents (voire un second ordinateur : /28/ TURNER & LEVY), soit une équipe de chronométreurs. Ces deux solutions sont inacceptables dans le cadre de ce travail et c'est pourquoi nous nous sommes tournés vers une possibilité existante.

Le système d'exploitation UNIX permet en effet à l'utilisateur de connaître le temps d'exécution de ses programmes (Cf. V.2 : Le montieur de test).

Lorsque l'on additionne ce temps d'exécution avec le temps de réflexion, on obtient le temps de cycle (Cf. ante). Ce dernier cependant est aléatoire.

Afin de garantir une parfaite reproductibilité de la charge, nous avons choisi de ne pas y inclure de temps de réflexion et de ne mesurer, à l'aide des commandes de UNIX, que les temps d'exécution des travaux.

I.5.2 Définition du but.

Il faut reconsidérer le but des mesures en fonction de la définition faite des performances.
Il ne nous appartient pas de considérer des limites acceptables du temps de réponse. Ceci ne serait en effet d'aucune utilité.

Dès lors, nous voulons mesurer et analyser les variations de la performance "temps de cycle restreint" en fonction de modifications planifiées de paramètres de base du système d'exploitation.

La suite de cet exposé nous permettra de voir comment cette performance sera mise en rapport avec les paramètres que nous considérerons.

I.5.3 Définition de l'outil.

L'outil de mesure que nous voulons construire devra donc nous permettre de collecter, au départ d'une charge donnée, les temps d'exécution des travaux qui la constituent.

Ceci doit se faire de manière itérative pour les valeurs différentes des paramètres.

De plus, l'outil doit permettre la modification des valeurs des paramètres d'après un plan préétabli et la régénération du système d'exploitation en tenant compte des nouvelles valeurs.

Enfin, les résultats des mesures doivent être traités et mis en forme afin d'éviter l'accumulation de gros volumes de données et de permettre l'analyse ultérieure.

Chapitre II

ANALYSE DE VARIANCE

Jean-Paul ADANS

Le problème qui nous préoccupe est d'étudier le comportement du système d'exploitation vis-à-vis de certaines grandeurs soumises à des variations. La valeur de ces grandeurs est fixée extérieurement et reste invariable, jusqu'à la prochaine variation.

Formellement, il s'agit de déterminer l'influence de certains paramètres - ou variables d'entrée - sur des entités mesurées - ou variables de sorties -. Ces dernières sont supposées à un certain point de vue, représentatives du comportement spécifique du système d'exploitation.

Dans le cas présent, deux éléments importants composent l'analyse statistique. D'une part, la planification des expériences, d'autre part, l'exploitation des résultats acquis au cours de ces expériences.

Ces deux parties sont étroitement liées. En effet, l'analyse doit être en mesure de répondre aux questions qui sont posées. Cela revient à dire que la planification doit produire les résultats nécessaires et suffisants. D'un autre côté, l'analyse doit pouvoir tirer un maximum de conclusions sur la base des informations produites par les expériences.

Les procédés que nous nous proposons ici de décrire entrent dans le cadre des tests dits non séquentiels, c'est-à-dire des expériences dont l'analyse est effectuée après chaque expérience.

Le but des différentes méthodes d'analyse de variance est de procurer un maximum de conclusions avec un minimum d'observations. Ceci parce que ces observations ne sont pas toujours faciles à obtenir, soit parce qu'elles découlent d'expériences dont la reproductibilité est réduite, ce qui est le cas par exemple en biologie, soit parce que ces expériences sont des tests de longue haleine et nécessitent un grand effort au point de vue temps et personnel. Dans la majorité des cas, ces expériences sont onéreuses et le budget alloué réduit à priori le nombre de tests possibles.

C'est dans ce but qu'il importe de considérer le lien étroit entre la planification des expériences et l'analyse proprement dite.

Avant d'aborder ces deux points, il faut définir ou rappeler certaines notions de base et conventions de terminologie propres à ces techniques.

II.1 Notions de base.

II.1.1 Modèle.

L'élément primordial dans l'utilisation des techniques d'analyse de variance est le modèle. Il s'agit d'une expression mathématique permettant d'appréhender la structure des données que l'on envisage d'étudier.

La forme générale du modèle peut être exprimée comme suit :

Valeur observée = paramètres représentant des effets significatifs
et variables aléatoires représentant des effets significatifs
et [variables aléatoires représentant des effets non significatifs (résidus)]

Par effets significatifs, nous entendons des effets qui peuvent de manière démontrable découler de changements dans les données. Les effets significatifs seront représentés par des FACTEURS. Par l'expérience, on sait que plus on augmente le nombre de facteurs représentant des effets significatifs, moins il y aura d'effets résiduels, bien que ces derniers ne disparaissent jamais.

De manière générale, les effets résiduels peuvent être pris en compte en introduisant des effets significatifs supplémentaires.

On espère cependant que ces effets résiduels soient négligeables. L'analyse de variance permet une étude malgré la présence de ces effets résiduels, qu'elle qu'en soit la source.

C'est le modèle qui permet de déterminer le type d'analyse qui convient, et qui n'est pas nécessairement unique.

II.1.2 Facteurs et niveaux.

Un facteur sera une qualité, une propriété vis-à-vis de laquelle les observations pourront être classifiées.

Chaque facteur présente différents NIVEAUX par rapport auxquels la classification peut être établie.

La structure de l'expérience reflète la manière dont les niveaux des facteurs sont combinés.

Si on utilise des paramètres pour représenter les effets des niveaux des facteurs, le modèle est dit paramétrique (ou fixe, ou est dit aléatoire (ou en composante de variance).

Etant donné la nature des facteurs entrant en ligne de compte dans le problème qui nous intéresse, nous ne parlerons dorénavant que des modèles paramétriques.

II.1.3 Variation résiduelle.

Il est intéressant de voir comment cette variation peut être représentée dans le modèle. Il est plausible de penser à une variable aléatoire. L'analyse de variance est basée sur certaines suppositions relatives à cette variable aléatoire :

(1) La moyenne de chaque variable aléatoire résiduelle est nulle.

Cette supposition traduit simplement le fait que chaque effet significatif peut être assigné à un paramètre ou à une variable aléatoire déterminée du modèle.

(2) Les variables aléatoires résiduelles sont mutuellement indépendantes.

Le sens de cette supposition est qu'il n'y a pas de lien entre différentes observations qui ne soit pris en compte par un effet significatif.

(3) Les variables aléatoires résiduelles ont toutes le même écart-type.

En fait, les techniques d'analyse de variance tentent de déterminer cet écart-type.

(4) Les variables aléatoires résiduelles sont distribuées normalement.

Ceci ne peut que constituer une approximation de la réalité. Dans certaines situations, elle est visiblement fausse. Mais cette supposition n'est pas indispensable à une grande partie de l'analyse de variance.

On voit que ces suppositions sont assez restrictives. Cependant, il est possible de mener à bien une analyse de variance en évitant certaines parties où ces assertions sont des conditions sine qua non. Nous verrons plus loin comment éviter les inconvénients résultant de ces limitations.

II.2 Classifications.

Nous avons vu que la classification était la façon dont les niveaux des différents paramètres étaient combinés entre eux.

Un premier exemple est celui de la classification simple.

II.2.1 Classification unique.

Dans l'exemple de la classification simple, on tente d'interpréter les observations en fonction des niveaux d'un seul facteur.

Sur une population de N individus, n_1 reçoivent le traitement 1, n_2 reçoivent le traitement 2, ... n_k le traitement k. Nous définissons le facteur "TRAITEMENT" dont les niveaux sont "traitement numéro t". A chaque niveau t est associé le groupe G_t, de cardinal n_t, des individus ayant reçu le traitement t.

On classe les observations dans un tableau de la forme du tableau II.1

Groupe	Observations
1	$\underline{x}_{1,1}, \underline{x}_{1,2}, \cdots \underline{x}_{1,n_1}$
2	$\underline{x}_{2,1}, \cdots \underline{x}_{2,n_2}$
.	
.	
.	
t	$\underline{x}_{t,1} \cdots \underline{x}_{t,n_t}$
.	
.	
k	$\underline{x}_{k,1} \cdots \underline{x}_{k,n_k}$

Tableau II.1 : Classification simple.

Les valeurs observées sont donc les $\underline{x}_{t,i}$ où $1 \leqslant t \leqslant k$ et $1 \leqslant i \leqslant n_t$. On observe de plus une variation résiduelle notée $\underline{z}_{t,i}$, dont les indices ont les mêmes limites que les $\underline{x}_{t,i}$.

Les suppositions (1) à (4) du paragraphe II.1.3 peuvent s'écrire :

(1) $E|\underline{z}_{t,i}| = 0 \quad \forall t, \forall i$

(2) $\underline{z}_{t,i}$ sont mutuellement indépendantes

(3) $\text{var}|\underline{z}_{t,i}| = \sigma^2 \quad \forall t, \forall i$

(4) $\underline{z}_{t,i}$ sont normalement distribuées.

Pour cette classification, le modèle s'écrit :

Valeur observée = [terme dû au niveau]
+ [résidu aléatoire]

soit

$$\underline{x}_{t,i} = G_t + \underline{z}_{t,i}$$

ou encore

$$\underline{x}_{t,i} = A + B_t + \underline{z}_{t,i}$$

où $G_t = A + B_t$ représente l'effet dû au niveau t : A est la moyenne, B_t la déviation au niveau t.

Comme on dispose dans le modèle transformé des k+1 paramètres $A, B_1, \ldots B_k$ pour représenter les k paramètres $G_1, \ldots G_k$, on aura la contrainte

$$\sum_{t=1}^{k} n_t B_t = 0$$

Ce qui exprime le fait que A est une moyenne des G_t. En fait

$$A = N^{-1} \sum_{t=1}^{k} n_t G_t$$

où $$N = \sum_{t=1}^{k} n_t$$

1. Calculons la moyenne de chaque groupe :

$$\underline{x}_{t,*} = n_t^{-1} \sum_{i=1}^{n_t} \underline{x}_{t,i}$$

de même

$$\underline{z}_{t,*} = n_t^{-1} \sum_{i=1}^{n_t} \underline{z}_{t,i}$$

d'après le modèle,

$$\underline{x}_{t,*} = A + B_t + \underline{z}_{t,*}$$

2. Calculons la moyenne générale :

$$\underline{x}_{**} = N^{-1} \sum_{t=1}^{k} n_t \underline{x}_{t,*}$$

et

$$\underline{z}_{**} = N^{-1} \sum_{t=1}^{k} n_t \underline{z}_{t,*}$$

on aura

$$\underline{x}_{*,*} = A + \underline{z}_{*,*}$$

Puisque

$$\sum_{t=1}^{k} n_t B_t = 0$$

3. Si on désire tester l'hypothèse $B_t=0$, on calculera

$$\underline{x}_{t,*} - \underline{x}_{*,*} = B_t + \underline{z}_{t,*} - \underline{z}_{*,*}$$

La moyenne de l'expression

$$\underline{x}_{t,*} - \underline{x}_{*,*}$$

sera égale à B_t tandis que son écart-type sera proportionnel à σ.

Pour estimer σ, il est nécessaire de tenir compte de tous les paramètres en construisant une fonction des observations qui puisse être exprimée uniquement en termes des $\underline{z}_{t,i}$.

Dans ce cas, ceci peut se faire en calculant les différences

$$\underline{x}_{t,i} - \underline{x}_{t,*} = \underline{z}_{t,i} - \underline{z}_{t,*}$$

Notons que, puisque :

$$\underline{x}_{t,i} - \underline{x}_{**} = (\underline{x}_{t,*} - \underline{x}_{**}) + (\underline{x}_{t,i} - \underline{x}_{t,*})$$

En effet, le calcul montre que le double produit s'annule en raison de l'hypothèse d'indépendance. Dès lors

$$\sum_{t=1}^{k} \sum_{i=1}^{n_t} (\underline{x}_{t,i} - \underline{x}_{*,*})^2 = \sum_{t=1}^{k} n_t (\underline{x}_{t,*} - \underline{x}_{*,*})^2 + \sum_{t=1}^{k} \sum_{i=1}^{n_t} (\underline{x}_{t,i} - \underline{x}_{t,*})^2$$

Les deux sommes de carrés du membre de droite sont appelées respectivement S.C. des écarts entre groupes et S.C. des écarts dans les groupes ou somme de carrés résiduelle.

Les sommes de carrés moyennes sont obtenues en divisant les S.C. par leur nombre de degrés de liberté.

Le calcul de l'espérance de ces S.C. permet d'évaluer la variation résiduelle.

La table d'analyse de variance ou (T.A.V.) résume ces calculs :

Origine	S.C.M.	Espérance
Variations entre groupes	$(k-1)^{-1} \sum_{t=1}^{k} n_t (\underline{x}_{t,*} - \underline{x}_{*,*})^2$	$\sigma^2 + (k-1)^{-1} \sum_{t=1}^{k} n_t B_t^2$
Variation résiduelle	$(N-k)^{-1} \sum_{t=1}^{k} \sum_{i=1}^{n_t} (x_{t,i} - \underline{x}_{t,*})^2$	σ^2

Tableau II.2 : T.A.V. de la classification unique.

On voit que pour interpréter cette T.A.V., il faut que l'assertion (3) soit valide. Cette supposition, que l'on nomme parfois homoscedasticité (cf /11/, JOHNSON & LEONE), est assez critique à ce niveau.

Si elle n'est pas garantie, les résultats obtenus par cette méthode seront vraisemblablement faux. Remarquons qu'il exite des procédés (tel le passage au logarithme ou à la racine carrée) qui permettent de mieux se rapprocher de cette restriction.

Par contre, l'hypothèse de normalité des variables aléatoires résiduelles peut être manipulée avec plus de souplesse. On peut démontrer qu'un certain écart à la normalité peut être toléré sans trop introduire de biais dans l'interprétation.

On peut cependant calculer les sommes de carrés d'une autre manière. Nous utiliserons alors les épreuves d'hypothèse pour l'interprétation des résultats.

Nous posons :

soient l le nombre de niveaux de facteur à étudier
n le nombre d'observations par cellules
(celui-ci peut être variable)

somme de carrés totale :

$$\sum_{t=1}^{l} \sum_{i=1}^{n} \underline{x}_{t,i} \quad \text{à} \quad l+n \text{ D.L.}$$

somme de carrés due aux niveaux du facteur :

$$\sum_{t=1}^{l} \underline{x}_{t,*}^{2} \quad \text{à} \quad n \text{ D.L.}$$

(somme de) carré par cellule :

$$\underline{x}_{**}^{2} \quad \text{à} \quad 1 \text{ D.L.}$$

Ces différentes expressions sont des variables aléatoires dont la distribution est une chi-carrée à respectivement l+n, n, 1 degrés de liberté.

Nous verrons plus loin, comment peut se faire l'interprétation à l'aide d'un test F.

II.2.2 Classifications multiples.

Lorsqu'on a deux ou plusieurs facteurs, il faut attacher de l'importance à la façon dont les niveaux des facteurs sont combinés.

Généralement, on distingue deux types de relations.

Dans la classification hiérarchique, à chaque niveau d'un facteur correspondent des sous-ensembles de niveaux des autres facteurs. Le tableau II.3 schématise une classification hiérarchique pour deux facteurs.

Facteur 1	Facteur 2	Observations
L1	1_1^1	$x_{1,1,1}$; $x_{1,1,2}$
	1_2^1	$x_{1,2,1}$; $x_{1,2,2}$; $x_{1,2,3}$
	1_3^1	---
L2	1_1^2	---
	1_2^2	---
L3	1_1^3	---
	1_2^3	---
	1_3^3	---
L4	1_1^4	---

Tableau II.3 : Classification hiérarchique à 2 facteurs.

Dans la classification croisée, chaque niveau d'un facteur peut être combiné avec tout niveau des autres facteurs. Le tableau II.4 donne un exemple de représentation d'une classification croisée à deux facteurs.

		Facteur 1				
		1.1	1.2	1.3	1.4	1.5
Facteur 2	2.1	---	---	---	---	---
	2.2	---	---	---	---	---
	2.3	---	---	---	---	---
	2.4	---	---	---	---	---

Tableau II.4 : Classification croisée à deux facteurs.

Nous avons choisi d'utiliser des classifications croisées pour notre étude. C'est en effet cette représentation qui s'adapte le mieux au problème des paramètres que nous nous sommes proposés d'étudier.

On utilise souvent une représentation matricielle pour les classifications croisées. Cette technique est également possible pour les classifications hiérarchiques mais conduit à des matrices creuses. La forme matricielle, si elle est facile à utiliser sur ordinateur, a cependant le désavantage d'être assez peu visuelle. Nous verrons dans l'exemple d'utilisation de la méthode comment nous avons visualisé les données.

A la différence du modèle à 1 facteur, les modèles à plusieurs facteurs ne peuvent se contenter de prendre en compte les effets dus aux niveaux des facteurs mais doivent également considérer les interactions entre niveaux de différents facteurs. Le nombre de termes dans l'expression du modèle croît dès lors comme des coefficients du binôme de Newton avec le nombre de facteurs.

Nous ne proposerons pas ici les démonstrations complètes pour des modèles à plusieurs facteurs. A titre d'exemple nous donnerons ici la définition des sommes de carrés relatives à un modèle à trois facteurs. La démarche est semblable pour les autres modèles.

Soient les facteurs B,C,D. Chacun de ces facteurs pouvant intervenir à n_B, n_C, n_D niveaux, le nombre total d'observations possibles est proportionnel au produit $n_B \times n_C \times n_D$, au nombre de répétitions près.

Soit n_R le nombre de répétitions par cellules. Le modèle complet s'écrit :

$$\begin{aligned} \underline{x}_{i,j,k,r} = A &+ B_i + C_j + D_k \\ &+ BC_{i,j} + BD_{i,k} + CD_{j,k} \\ &+ BCD_{i,j,k} \\ &+ \underline{z}_{i,j,k,r} \end{aligned}$$

où $1 \leqslant i \leqslant n_B$

$1 \leqslant j \leqslant n_C$

$1 \leqslant k \leqslant n_D$

$1 \leqslant r \leqslant n_R$

avec les contraintes

$$\sum_{i=1}^{n_B} B_i = 0$$

$$\sum_{j=1}^{n_C} C_j = 0$$

$$\sum_{k=1}^{n_D} D_k = 0$$

$$\sum_{i=1}^{n_B} BC_{ij} = 0, \forall j \qquad \sum_{j=1}^{n_C} BC_{ij} = 0, \forall i$$

$$\sum_{i=1}^{n_B} BD_{ik} = 0, \forall k \qquad \sum_{k=1}^{n_D} BD_{ik} = 0, \forall i$$

$$\sum_{j=1}^{n_C} CD_{jk} = 0, \forall k \qquad \sum_{k=1}^{n_D} CD_{jk} = 0, \forall j$$

On définira les sommes de carrés comme dans le cas du modèle à un facteur, c'est-à-dire en sommant les carrés des différentes moyennes qu'il est possible d'établir dans ce modèle.

Remarque : il est important de connaître l'influence du nombre de répétitions par cellule : si $n_R = 1$, la valeur que l'on calculera pour la somme des carrés à l'intérieur des cellules n'a pas de sens. Autrement dit, il ne serait pas possible d'estimer la variance résiduelle des observations par cellules.

Dans ce cas, certaines interactions ne pourront pas être appréhendées. Dans la technique que nous avons choisie, c'est l'interaction de plus haut niveau qui ne pourra pas être évaluée. Pour l'exemple, l'effet dû à l'interaction de niveau 3, c'est-à-dire $BCD_{i,j,k}$ se confondra avec la variation résiduelle.

Lorsqu'on estime que cet effet a beaucoup d'importance, on prendra soin d'exécuter les expériences en plusieurs exemplaires (n_R=2,3,...) (Cf II.4 Interprétation de l'analyse de variance).

Nous avons ainsi défini la manière de calculer les différentes sommes de carrés pour les différents modèles. En annexe, le lecteur trouvera l'expression des sommes de carrés pour des modèles paramétriques croisés de un à cinq facteurs.

La table d'analyse de variance permet de fournir les résultats de l'analyse sous une forme visuelle. Elle reprend la définition des effets, c'est-à-dire à quel facteur ou combinaison de facteurs tel effet peut être attribué.

Elle donne la somme de carrés pour chaque effet ainsi que le nombre de degrés de liberté y afférant. La somme de carrés moyenne est obtenue en divisant la somme des carrés par le nombre de degrés de liberté.

Dans un calcul manuel, on peut restreindre les opérations à l'évaluation de certaines sommes de carrés que l'on juge à priori intéressantes. Dans un calcul sur ordinateur,

on peut se permettre de calculer toutes les sommes de carrés et ainsi être sûr de pouvoir tester n'importe quelle hypothèse.

II.3 Plans d'expériences.

II.3.1 Plans complets.

Comme nous l'avons vu, dans une classification croisée, tous les niveaux de chaque paramètres entrent en combinaison avec tous les niveaux des autres paramètres. Dès lors, le modèle reflète non seulement les effets dûs aux niveaux des facteurs, mais également les effets dûs aux différentes interactions.

Les plans d'expérience pour des classifications de ce genre doivent donc tenir compte de toutes ces combinaisons et sont par conséquent, souvent très volumineux une fois que l'on dépasse deux facteurs. Par exemple, si on considère quatre facteurs pouvant chacun intervenir à trois niveaux, il faut au minimum quatre-vingt une expériences.

La planification de ces expériences est simple : il suffit d'écrire toutes les combinaisons possibles de tous les niveaux de tous les facteurs. Ensuite, il convient d'introduire un aléas en permutant au hasard ces combinaisons, si le sujet d'expérience risque de garder une rémanence d'une épreuve à l'autre.

Dans notre cas, cet élément aléatoire n'a pas été nécessaire, étant donné que le système d'exploitation est régénéré avant chaque expérience (cf. Chap. VI).

II.3.2 Plans incomplets.

Nous l'avons dit, des plans croisés complets deviennent rapidement très volumineux avec l'accroissement du nombre de paramètres.

Deux alternatives sont possibles : si les expériences sont relativement courtes et peu coûteuses, il est possible jusqu'à un certain point d'exécuter des plans complets. Si par contre, elles sont longues, il faut renoncer à investiguer toutes les combinaisons possibles, soit en réduisant les nombres de niveaux à priori, soit en adoptant des techniques de planifications plus complexes.

Ces techniques sont connues sous le nom de plans d'expérience incomplets. Comme le nom l'indique, elles réalisent une planification où une partie seulement des combinaisons possibles est effectivement utilisée.

Suivant le type de planification incomplète, elles réduisent le nombre des expériences d'un facteur 2,3 ou plus.

L'inconvénient majeur de ces techniques est évident. La réduction du nombre d'épreuves va de pair avec une diminution des informations produites par l'analyse. Il faut être conscient de cette perte, et souvent ces techniques nécessitent une connaissance à priori du phénomène à analyser, ce qui permet d'éliminer des informations que l'on peut supposer d'avance être sans intérêt. Ceci présente bien sûr un certain danger et l'utilisation de telles méthodes doit se faire avec la plus grande circonspection.

La littérature renseigne différentes techniques de planification, toutes plus ou moins adaptées à un certain type d'expérience. Ce domaine est surtout étudié à la demande d'autres branches de la science, lorsque le besoin s'en fait sentir.

Citons en particulier la méthode des blocs aléatoires incomplets, où les expériences sont groupées en fonc-

tion des disponibilités du sujet d'expérience (agronomie, biologie) et la méthode des plans fractionnaires.

Cette dernière paraît le mieux adaptée au cas présent. A l'aide d'un facteur discriminant choisi parmi l'ensemble des effets possibles, on définit deux sous-ensembles d'effets. Seuls les effets de l'un des sous-ensembles peuvent être étudiés, ceux du second étant confondus avec ceux du premier. Ici encore, il faut être conscient de la perte d'informations et le facteur discriminant ne peut être choisi qu'en connaissance de cause.
Le tableau V.4 donne un exemple de plan fractionnaire, où on a scindé les effets d'un plan complet à 5 facteurs ABCDE, chacun à deux niveaux, à l'aide du contraste définissant $A_1\ B_1\ C_1\ D_1\ E_1$

Effet	Effet confondu
$A_o\ B_o\ C_o\ D_o\ E_o$	$A_1\ B_1\ C_1\ D_1\ E_1$
$A_1\ B_o\ C_o\ D_o\ E_o$	$A_o\ B_1\ C_1\ D_1\ E_1$
$A_o\ B_1\ C_o\ D_o\ E_o$	$A_1\ B_o\ C_1\ D_1\ E_1$
$A_o\ B_o\ C_1\ D_o\ E_o$	$A_1\ B_1\ C_o\ D_1\ E_1$
$A_o\ B_o\ C_o\ D_1\ E_o$	$A_1\ B_1\ C_1\ D_o\ E_1$
$A_o\ B_o\ C_o\ D_o\ E_1$	$A_1\ B_1\ C_1\ D_1\ E_o$
$A_1\ B_1\ C_o\ D_o\ E_o$	$A_o\ B_o\ C_1\ D_1\ E_1$
$A_1\ B_o\ C_1\ D_o\ E_o$	$A_o\ B_1\ C_o\ D_1\ E_1$
$A_1\ B_o\ C_o\ D_1\ E_o$	$A_o\ B_1\ C_1\ D_o\ E_1$
$A_1\ B_o\ C_o\ D_o\ E_1$	$A_o\ B_1\ C_1\ D_1\ E_o$
$A_o\ B_1\ C_1\ D_o\ E_o$	$A_1\ B_o\ C_o\ D_1\ E_1$
$A_o\ B_1\ C_o\ D_1\ E_o$	$A_1\ B_o\ C_1\ D_o\ E_1$
$A_o\ B_1\ C_o\ D_o\ E_1$	$A_1\ B_o\ C_1\ D_1\ E_o$
$A_o\ B_o\ C_1\ D_1\ E_o$	$A_1\ B_1\ C_o\ D_o\ E_1$
$A_o\ B_o\ C_1\ D_o\ E_1$	$A_1\ B_1\ C_o\ D_1\ E_o$
$A_o\ B_o\ C_o\ D_1\ E_1$	$A_1\ B_1\ C_1\ D_o\ E_o$

Tableau II.4: Plan fractionnaire 2^5
$A_1\ B_1\ C_1\ D_1\ E_1$ pris comme effet discriminant.

On voit qu'on obtient les effets confondus en multipliant modulo 2 les effets désirés par le contraste définissant. La taille du plan d'expérience est ainsi réduite de moitié.

Plusieurs ouvrages donnent pour cette technique des tables déterminant les expériences qui sont à faire (par exemple, /11/ JOHNSON & LEONE).

L'analyse de variance se réduit alors au calcul des sommes de carrés définies par ces tables.

II.4 Interprétation de l'analyse de variance.

II.4.1 T.A.V.

Nous avons décrit plus haut comment elle se construit.

Moyennant l'acceptation de l'hypothèse de homoscedasticité, l'interprétation peut se faire par l'expression de l'espérance des sommes de carrés.

Nous avons choisi d'utiliser des épreuves d'hypothèse et c'est pourquoi nous avons redéfini les sommes de carrés.

II.4.2 Test F.

Lorsque l'on dispose de ces sommes de carrés, on peut tester certaines hypothèses, soit par rapport au modèle le plus général, soit par rapport à un autre modèle.

Le modèle le plus général est celui qui admet que la classification est unique, c'est-à-dire qu'à chaque cellule peut être associé un niveau d'un seul facteur.

Pour réaliser une telle épreuve, on calculera le quotient

$$\underline{r}_H = \frac{SCG - SCH \;/\; DL_G - DL_H}{SCE \;/\; DL_E}$$

où

SCG est la S.C. du modèle le plus général (ou celle du modèle par rapport auquel on veut tester).

SCH la S.C. du modèle sous épreuve

SCE la S.C. des écarts du modèle le plus général par rapport à la S.C. totale

DL_G, DL_H, DL_E les nombres de degrés de liberté respectivement des S.C. citées

Ce quotient est une variable aléatoire de loi de distribution F (de Snedecor) à $(DL_G - DL_H)$ et DL_E degrés de liberté.

Il sera dès lors comparé au quantile de cette distribution pour le niveau d'incertitude voulu (en général, les tables donnent les valeurs pour 1% et 5% d'incertitude)

Si le rapport excède la valeur du quantile, l'hypothèse sous épreuve devra être rejetée. Sinon elle ne peut être rejetée et doit être provisoirement admise.

II.4.3. Cas particulier : une répétition.

Nous avons signalé ce cas particulier. Le modèle le plus général ne pourra pas être utilisé comme référence. En effet, si une seule répétition a été faite, la S.C. de ce modèle sera égale à la S.C. totale, et le dénominateur du quotient $\underline{r}_H$ sera égale à $\frac{0}{0}$!

Il faut alors utiliser un autre modèle. En général, on utilisera le modèle additif, contenant tous les

effets et toutes les interactions sauf celle du niveau le plus élevé.

On calcule la S.C. de ce modèle par une formule donnée en annexe.

Chapitre III

L'ENVIRONNEMENT EXPERIMENTAL

Jean VERCHEVAL

III.1 UNIX.

UNIX est un système d'exploitation interactif, multiutilisateur pour les ordinateurs du haut de la gamme DEC PDP 11 ainsi que pour le INDERDATA 8/32.

Il offre des caractéristiques qui ne sont pas courantes même pour des systèmes plus importants, comme

- une organisation hiérarchique des fichiers, permettant des volumes démontables;
- des processus d'entrée/sortie compatibles; pour les fichiers, périphériques et entre processus;
- la possibilité d'initier des processus asynchrones;
- le langage de commande (SHELL), sélectif par utilisateur;
- plus de 100 sous-systèmes incluant une douzaine de langages;
- une très grande portabilité.

(tiré de /22/, RITCHIE & THOMPSON).

UNIX est un système très répandu dans les milieux universitaires et de nombreux logiciels ont été développés par ces centres. Outre des compilateurs et interpréteurs pour une douzaine de langages (PASCAL, SNOBOL, APL, ALGOL 68, M6, TMG, ...) sont disponibles des logiciels très divers tel par exemple un assembleur pour microprocesseur (8080asm développé par U.C.L.A.) etc...

De plus, chaque université apporte des modifications au noyau de UNIX. Ainsi, il existe une version du système pour un microprocesseur DEC LSI-11 équipé de 20 K-mots de mémoire centrale et de disques souples pour mémoire auxiliaire.

La version standard de UNIX, dont nous avons fait usage, comprend outre un compilateur C, un compilateur fortran, un assembleur, un interpréteur BASIC, des programmes de chargement et de diagnostic, ainsi que un macro-assembleur. Elle permet la mise en forme de textes, la sortie graphique de dessins et de symboles mathématiques, et, avec

le périphérique approprié, la préparation de négatifs pour l'impression offset.

Une particularité est également le traitement de grammaires formelles pour la réalisation de compilateur. (voir /27/, THOMPSON & RITCHIE).

III.1.1 Le noyau.

Le noyau du système UNIX est décrit dans le langage de programmation de haut niveau C. (à 95% : 5% sont écrits en assembleur). Nous ne décrirons pas en détail ce langage, mais disons qu'il permet la manipulation de structures de données, de bits et la définition de nouveaux types de variables. Il présente plusieurs formes de contrôle de boucle, ainsi que des facilités d'initialisation et de gestion de processus asynchrones.

La taille du noyau est relativement petite (10.000 lignes de code source) et son étude peut être poussée assez loin.

Il est donc aisé d'apporter des modifications au système, puisque le code source est directement accessible par fichier. Il suffit de compiler le système pour le régénérer. Il est tout aussi facile d'ajouter des commandes au SHELL, qui sont des programmes banalisés.

III.1.2 Les fichiers.

Ceux-ci sont organisés de manière hiérarchique en directoires.

Ils peuvent être munis de protections d'accès simples mais généralement suffisantes.

Les fichiers se subdivisent en :

- directoires, qui fournissent des chemins d'accès aux fichiers qui y figurent;
- fichiers spéciaux, représentant les périphériques;
- fichiers, au sens classique du terme.

Les mêmes primitives permettent donc de traiter les périphériques et les fichiers normaux.
En particulier, les terminaux sont facilement accessibles.

Aucune structuration classique n'est prévue par le système (méthodes d'accès séquentielle, séquentielle indexée, aléatoire, ...). Un fichier est uniquement considéré comme une suite de caractères, éventuellement découpée par des caractères de passage à la ligne pour les fichiers de texte.

Il appartient à l'utilisateur de manipuler l'information contenue dans un fichier (au sens large), comme il l'entend.

La structure de directoires permet de définir un chemin d'accès aux fichiers. La racine de l'arborescence ("root") est notée "/". Un fichier sera défini par sa position relative soit à la racine (ex : "/usr/sys/ken/main.c") soit au directoire "courant" (celui dans lequel l'utilisateur se trouve. Par exemple, si le directoire courant est "/usr/sys", le même fichier sera référencé par "ken/main.c").

Par convention, le caractère "." dénote le directoire courant. (si "/usr/sys" est le directoire courant, "ken/main.c" et "./ken/main.c"). Les caractères ".." représentent le directoire "parent", c'est-à-dire celui directement supérieur dans l'arborescence (si "." est "/usr/sys/ken", "../conf/1.s" représente "/usr/sys/conf/1.s").

Deux fichiers qui portent le même nom ne sont identiques que s'ils se trouvent dans le même directoire, ou si un lien ("link") a été explicitement établi entre eux. Ainsi, "/usr/sys/tune.h" est différent de "/apm/tune.h".

Un postfixe est souvent ajouté au nom d'un fichier pour identifier son contenu :

XXXX.c représente un fichier contenant un programme source en C

XXXX.f représente un programme source en Fortran

XXXX.s représente un programme source en assembleur

XXXX.o représente un programme en version exécutable

XXXX.h est souvent utilisé pour un fichier ne contenant que des déclarations pour un programme en C.

Les périphériques sont représentés par des fichiers se trouvant dans le directoire "/dev". Ainsi "/dev/mt"représente une unité à bande magnétique.

Lorsqu'un support externe contient une organisation de fichiers, on peut y accéder en "montant" le volume.

Le système d'exploitation met à la disposition de l'utilisateur des primitives pour créer, ouvrir, lire, écrire, parcourir et détruire un fichier. Des protections d'accès garantissent la sécurité tant au niveau des directoires que des fichiers. Ces protections sont pour la lecture, l'écriture, l'exécution et concernent le propriétaire du fichier, les utilisateurs du même groupe et les autres utilisateurs.

III.1.3 Utilisateurs.

Ceux-ci sont organisés en groupes bénéficiant des mêmes possibilités.

A chaque utilisateur est associé un directoire courant. Lorsqu'une identification est correcte, l'utilisateur se trouve dans ce directoire. A son terminal est associée une image (définie dans /22/ comme un environnement d'exécution) de l'interpréteur du langage le commande ("SHELL").

III.1.4 Processus.

Un processus est l'exécution d'une image. A chaque terminal est associé un processus "SHELL".

Deux processus sont initiés au chargement du système. L'un réalisant la gestion des lignes, l'autre assurant la gestion des processus (exécution du noyau du système d'exploitation).

Lors de l'introduction d'une commande à un terminal actif, un nouveau processus est créé et le "SHELL" attend la fin de son exécution.

Lorsqu'une commande est suivie du caractère "&", le "SHELL" n'attend pas la fin de l'exécution de cette commande et permet ainsi l'initiation de plusieurs processus asynchrones.

III.2 Le "SHELL". (1)

Un "SHELL" est un programme qui interprète les commandes de UNIX. Quand un utilisateur est"connecté", il communique avec le système en tapant des commandes. Celles-ci sont interprétées ligne par ligne, sont écrites dans un langage de commande destiné à exécuter des programmes existants, à communiquer avec d'autres, avec des fichiers, avec des supports physiques, et ce, en utilisant les constructions des fonctions primitives (cfr II.2.1 Les primitives).

Le langage de commandes est interprété par le programme "sh". Ce dernier envoie le caractère "%" sur le terminal, ensuite il lit la commande écrite par l'utilisateur, le décode en ses composants (le (s) nom (s) du (des) programmes (s) à exécuter et ses (leurs) arguments), rend possible toute facilité de communication ("pipe" - Cfr II. 2.2 Les primitives) demandée entre les programmes de la commande, exécute le(s) programmes(s).

(1) Cfr /6/ COLOURIS G., /27/ THOMSON K.D. et RITCHIE D.M.

Le "SHELL" n'est pas une primitive. Il traite tout son travail sans recourir à d'autres possibilités que le langage C et les primitives accessibles à partir de programmes écrits en C. Une conséquence intéressante est que chaque utilisateur peut implémenter son propre interpréteur de commande ou plus simplement ajouter ou modifier des commandes. C'est grâce à cette caractéristique que la commande "reboot" du chargement automatique, a pu être "aisément" implémentée.

II.2.1 Les Primitives.

A. Au niveau fichier.

A chaque fichier sont associés, au minimum, un nom et un jeu de caractéristiques sur l'état du fichier. Il existe six primitives pour manipuler le nom et l'état du fichier.

"creat" est employé pour créer un nouveau fichier (initialement vide)

"chown" modifie le statut de possession du fichier

"chmod" modifie la (les) permission(s) read-write-execute

"link" donne un nom supplémentaire (un synonyme) au fichier

"unlink" élimine un nom de fichier (pas le fichier sauf si on détruit son unique nom

"stat" donne l'état du fichier

Il en existe cinq autres pour réaliser des transferts entre programmes fichiers et devices. Il s'agit de : "open", "close", "read", write", "seek". Leur emploi est évident : ouverture, écriture, d'un fichier.

Cependant lors de l'usage d'un "open", un nombre appelé "file descriptor" est communiqué au programme (si la demande est explicite). Ce nombre servira d'identificateur externe du fichier ainsi ouvert et sera nécessairement employé

dans les primitives "read" et "write". Sa valeur est de peu d'importance, disons simplement qu'elle n'est pas intrinsèquement liée au fichier. Elle peut varier, pour un même fichier, d'un programme à l'autre.

Les primitives "read" et "write" permettent à l'utilisateur de spécifier le nombre de caractères à lire ou à écrire. Il n'y a qu'une différence entre un programme qui lit un fichier caractère par caractère et un autre qui le fait par 1000 caractères : c'est la vitesse d'exécution. Son optimum tant en lecture qu'en écriture se situe au niveau des transferts de 512 caractères.

B. Niveau communication - "pipes".

Une "pipe" est une primitive qui est utilisée pour transmettre des séquences de caractères entre des processus employant les primitives "read" et "write" comme pour des fichiers.

L'opérateur "^" employé avec des commandes crée une "pipe".

Exemple : a ^ b

Exécute les programmes a et b en concurrence, avec une "pipe" reliant la sortie standard de a à l'entrée standard de b.

N.B.: L'entrée standard est désignée par le "file descriptor 0" qui en général est associé au clavier terminal. Quant à la sortie standard elle est désignée par "1", en général associé au vidéo du terminal.

C. Niveau processus.

Définissons ici un processus comme étant un programme en état d'être exécuté. Toute exécution de programme

se fait par le "SHELL". En effet une commande est un programme à exécuter dont les arguments peuvent varier selon son écriture. Ainsi l'exécution d'un programme fortran doit être demandée par une commande bien précise mais ses arguments diffèrent d'un programme fortran à l'autre. A la limite on peut dire que le programme chargé de l'exécution de programmes fortran a pour argument le programme fortran lui-même.

Chaque fois qu'un programme doit être exécuté, un nouveau processus est créé et existera jusqu'à la fin du programme. Cette création de processus se fait par l'emploi d'une primitive : "fork". Elle a pour effet de créer un nouveau processus semblable à celui qui l'a utilisée, avec toutes ses caractéristiques sauf aune :"fork()" renvoie un numéro d'identification (différent de zéro) pour le nouveau processus dans le processus appelant (le processus parent), et zéro dans le nouveau processus (le processus enfant). Cela revient à dire que si dans un programme on a :

```
n = fork();
imprimer n;
```

n vaudra o s'il s'agit de l'exécution d'un processus enfant et n≠o si c'est le processus parent qui s'exécute. On peut leur faire subir des traitements différents au sein du même code par l'emploi d'une instruction conditionnelle :

```
if n≠o then<instruction>
```

Lorsque le nouveau processus est créé, il s'exécutera en concurrence avec le parent (c'est-à-dire chacun indépendamment), à une vitesse déterminée par le gérant des processus.

Une autre primitive "exec" permet à un processus de changer son programme en chargeant un nouveau à partir d'un fichier.

Ainsi le "SHELL" interprète une commande par un "fork". Un nouveau processus est ainsi créé. Pour la plupart

des commandes, suit un appel à "exec" qui va transformer le processus afin de tenir compte des arguments de la commande. La primitive "exec" joue donc un rôle important dans le système puisqu'en fait c'est elle qui caractérise le processus.

Le "fork" crée toujours le même processus pour une commande et "exec" tient compte des arguments de la commande envoyée par un utilisateur.

Cependant des commandes comme "chdir" sont traitées directement sans recourir à un "fork" et à un "exec".

D. Niveau synchronisation.

La communication entre programmes demande souvent un degré de synchronisation. C'est le rôle des primitives "sleep" et "wait", "signal" et "kill".

"sleep" provoque l'interruption du processus pendant le temps spécifié.

"wait" est utilisé chaque fois qu'un processus ne doit pas être exécuté jusqu'à ce que ses processus enfants aient terminé.

"kill" transmet un signal d'un processus à un autre dont le numéro d'identification est connu du premier. Il existe 16 valeurs de signaux. Si le processus receveur n'a pas pris une action prioritaire pour recevoir le type particulier de signal envoyé, il est détruit aussitôt le signal reçu.

"Signal" est une primitive utilisée comme action préparatoire avant un "kill". Elle permet à un programme de désigner une procédure pour chaque type de signal. Ainsi chaque fois qu'un signal du type donné est reçu, la procédure est appelée automatiquement, quelque soit ce qui a été fait avant par le processus.

III.2.2 Quelques caractéristiques.

Le langage de commande est assez étoffé et il n'est pas possible même simplement le survoler. Nous expliquons ici quelques commandes qui jouent un rôle dans la charge et dans le moniteur de test.

-sh

Il est possible d'exécuter toute une suite de commandes placées dans un fichier en employant "sh".

sh nom-de-fichier

exécute toutes les commandes contenues dans nom-de-fichier sauf si une de celles-ci ne peut être exécutée convenablement pour une raison ou l'autre.

Exemple : dans fichier 1, nous avons la succession de commandes (avec les résultats) :

- compiler un programme (des erreurs apparaissent),
- modifier le programme (pour qu'il soit compilable sans erreur),
- compiler à nouveau (plus d'erreur).

Sans précaution particulière, lors du lancement de "sh fichier 1", le déroulement des opérations est le suivant :

- compilation du programme,
- des erreurs sont détectées,
- l'exécution des autres commandes ne se fait pas.

Pour contrer cet effet, on peut recourir à une deuxième commande "sh" et à 2 fichiers. Dans fichier 1, on écrit :

- sh fichier 2,
- modifier le programme,
- compiler à nouveau.

Dans fichier 2, il n'y a que la commande :

- compiler le programme.

On lance la commande "sh fichier 1". Les effets produits sont :

- exécuter les commandes de fichier 2, c'est-à-dire

- compiler le programme,
- des erreurs apparaissent,
- l'exécution de "sh fichier 2" se termine,

- modifier le programme,
- compiler à nouveau,
- fin de l'exécution de "sh fichier 1" car il n'y a plus d'autre commande.

-&

Ce signe permet d'exécuter simultanément une ou plusieurs commandes, et par extension un ou plusieurs fichiers de commandes.

<commande 1> & <commande 2>

- L'indirection.

Elle permet de dévier les messages de retour de la sortie standard - le terminal sur lequel on travaille - vers un fichier quelconque

> : indirection qui a pour effet d'enlever et de détruire tout ce qui se trouve initialement dans le fichier pour y mettre le message qu'elle désire;

>> : indirection qui ajoute au contenu du fichier le message dévié.

N.B. UNIX généralise la notion de fichier au point que tout périphérique est considéré comme un fichier.

- chdir

Permet de changer de directoire pour y travailler avec les fichiers qu'il contient (cfr III.3.4.4 Directoires de fichier et fichiers de directoire).

- Time

C'est une commande qui fournit trois temps pour une autre commande qui s'exécute:

time <commande>

donne les trois temps suivants :

"real" : temps écoulé pendant la commande,
"user" : temps passé dans le système,
"sys" : temps passé pour l'exécution de la commande.

- sleep

C'est une commande qui suspend l'exécution pendans un certain temps:

sleep N

suspend l'exécution pendant N secondes.

III.3 Le système d'exploitation UNIX (1).

Le but initial de cette partie était de résumer les principaux mécanismes et les propriétés les plus marquantes de UNIX. Mais nous nous sommes aperçus que ce n'était possible sans rentrer dans de nombreux détails. Aussi, avons-nous limité nos ambitions à donner du mieux possible une introduction au système d'exploitation UNIX.

Nous nous basons sur l'explication de dix paramètres du système pour montrer certaines généralités.

D'autres notions sont exposées pour faciliter l'interprétation de nos mesures.

---

(1) Inspiré de /18/ LIONS J., /25/ ROWSON J, /9/ DIGITAL EQUIPMENT CORPORATION.

III.3.1 La mémoire.

Les ordinateurs DEC PDP 11/45 et PDP 11/70 tournent sur trois modes. Le mode utilisateur est utilisé quand le processus actif est celui d'un utilisateur, le mode noyau quand il est spécifique au système, le mode superviseur est utilisé pour gérer la multiprogrammation. Cependant nous ne faisons que mentionner ce dernier car UNIX ne l'emploie pas.

Chaque mode peut avoir deux espaces. L'espace "i" désigne les adresses utilisant le compteur ordinal ("P-count") et l'espace "d" les autres adresses.

Chaque espace possède son propre jeu de registres de segmentation, ce qui permet de doubler l'espace virtuel.

Par la suite, nous utilisons l'expression espace noyau(ou utilisateur) sans distinguer s'il s'agit de l'espace "i" ou "d" de ce mode par souci de clarté. De plus le fait de doubler l'espace virtuel ne peut jouer un rôle dans nos mesures de performance.

Les programmes peuvent adresser jusqu'à 64 KBytes avec des adresses de 16 bits. On peut aller jusqu'à 256KBytes avec un mécanisme d'adresses virtuelles de 16 bits équivalentes à des adresses physiques de 18 bits. Ce mécanisme se nomme "Memory Management Unit" (MMU).

La mémoire est divisée en pages virtuelles de 8 KBytes. Ces pages ont la particularité de pouvoir comporter deux parties. Une de celles-ci est déclarée contenir les adresses virtuelles valides. Toute tentative d'accès à l'autre partie produit une interruption interne hardware. Cette découpe permet d'économiser de la mémoire.

II.3.1.1 Adressage.

A. Conditions initiales.

Quand le système démarre, l'instruction RESET a pour effet de réinitialiser tous les supports sur l'UNIBUS, de mettre tous les registres de segmentation à zéro, de mettre le processeur en mode noyau.

Les adresses virtuelles dans ce mode de 0 à 56 KBytes sont identiques aux adresses physiques. Cependant les adresses virtuelles de la plus haute page sont translatées en les adresses physiques de la plus haute page c'est-à-dire la huitième page virtuelle équivaut à la dernière page physique soit les adresses virtuelles 56 KBytes à 64 KBytes en les adresses physiques 248Kbytes à 256 KBytes.

Cette dernière page contient uniquement des registres spéciaux associés au processeur et aux supports périphériques. En sacrifiant une page, les constructeurs permettent d'accéder à ces registres sans instruction spéciale.

B. Construction d'une adresse physique.

L'outil de base est une paire de registres : les registres de segmentation. Un jeu comporte huit paires et il existe un jeu par espace "i" et un par espace "d" d'un même mode.

Une paire se compose d'un "Page Address Register" (PAR) et d'un "Page Descriptor Register" (PDR).

Interprétation d'une adresse virtuelle.

Une adresse virtuelle comporte 16 bits décrits comme suit :

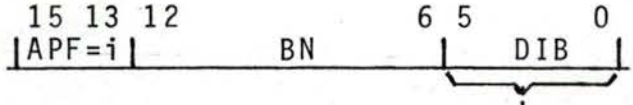
- les bits 0 à 12 forment le "Displacement Field" (DF) qui contient une adresse relative au début de page. Il se décompose en le "Block Number" (BN) (bit 6 à 12) qui fournit le numéro de bloc à l'intérieur de la page courante et le "Displacement In Block" (DIB) (bit 0 à 5) qui donne le déplacement à l'intérieur du bloc BN.
- les bits 13 à 15 déterminent l' "Active Page Field" (APF) ou le registre PAR à utiliser pour la conversion.

Interprétation du registre PAR.

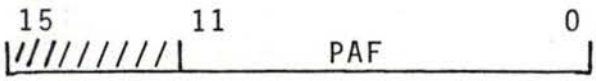
Ce registre contient 16 bits dont les 12 premiers donnent le "Page Address Field" (PAF) qui est l'adresse du début de la page mémoire que le PAR spécifie. Cette adresse est donnée sous la forme d'un numéro de bloc dans la mémoire physique. Un bloc vaut 32 mots.

Construction.

On a l'adresse virtuelle suivante :



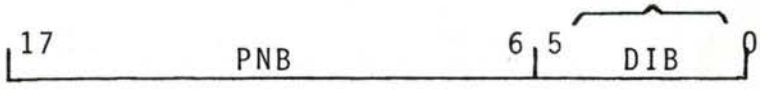
donne le PAR n° i soit :



ces 6 bits se placent à la suite du PNB

on calcule le "Physical Number Block" (PNB)
PNB = PAF + BN

Et on obtient l'adresse physique de 18 bits



C. Remargues.

C'est le "Processor Status Word" qui indique le mode courant. Ainsi le processeur sait quel jeu de registres de segmentation prendre. De plus un registre d'état permet de sélectionner soit la mise hors usage de l'espace "d" pour translater toutes les références dans le "i", soit l'utilisation des deux espaces. Ainsi quand le processeur traite une adresse de type "d", il sait quel jeu utiliser.

Chaque jeu couvre 8x8 KBytes soit 64 KBytes. Utiliser deux espaces double l'espace adressable virtuel.

Cependant dans nos mesures, la mémoire physique est limitée à au plus 128 KBytes. Or l'espace "i" d'un mode couvre 64 KBytes. Les espaces "i" des deux modes sont donc suffisants pour couvrir toute la mémoire physique. C'est pourquoi nous ne parlons plus, par la suite, des espaces "i" et "d" mais d'un espace par mode.

Toute page virtuelle a un PAR et un PDR associés. Ces registres se nomment registres de segmentation ou "Active Page Register" (APR).

III.3.1.2 Les registres PAR du mode noyau.

Les registres PAR 0 à 5 sont initialisés pour translater les adresses virtuelles du noyau 0 à 48 KBytes en les adresses physiques 0 à 48 KBytes, ce qui correspond aux six premières pages de la mémoire centrale où UNIX réside en permanence. C'est toujours vrai si la longueur combinée du code UNIX et des données est inférieure à 48 KBytes !

Le registre PAR7 translate les adresses virtuelles 56 à 64 KBytes en les adresses physiques 248 à 256 KBytes.

Le registre PAR6est le seul qui change dynamiquement. En général il pointe vers la part de la mémoire physique qui contient le segment de données du processus courant en exécution (cfr III.2.1.1 Description des composants d'une image d'un processus).

III.3.1.3 La page.

Le système adresse la mémoire principale par bloc de 32 mots. Cependant il utilise les registres de segmentation comme descripteurs de page. C'est en quelque sorte des pages virtuelles.

Une page s'étend sur 8 KBytes. Le PAR donne l'adresse de début de page. Le "Page Descripter Register" (PDR) contient les informations sur cette page. Il fournit des indications sur les accès permis (lecture, lecture et écriture, ...), sur les modifications éventuelles de l'état de la page (on a écrit, ...).

Une page se découpe toujours en deux parties (dont une peut être nulle). La partie valable est celle pour laquelle les références à des adresses sont permises. Toute tentative pour adresser une zone de l'autre partie provoque une interruption interne. Les indications sur la grandeur des deux parties, sur les adresses valables se trouvent aussi dans le PDR. Elles se nomment partie haute et partie basse.

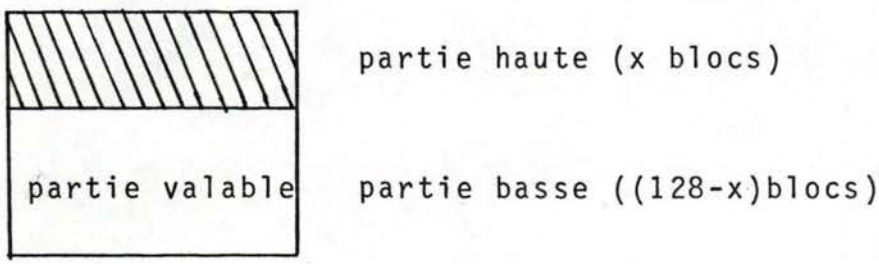


Fig. III.1 Découpe d'une page

La partie valable doit comporter au moins un bloc et au plus 128 blocs. Il est possible de l'agrandir (jusqu'à 128 blocs).

L'utilité de cette découpe peut se montrer par l'exemple de l'adressage du segment de donnée d'un processus.

En bref, ce segment se divise en deux éléments dont un est référencé par les registres de segmentation 6 du mode noyau, l'autre par les registres du mode utilisateur. Le premier est toujours grand de 1024 Bytes, l'autre peut être de taille variable (cfr fig. III.2).

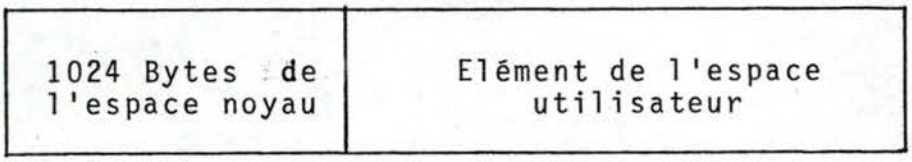


Fig. III.2 Le segment de donnée

La page 6 de l'espace noyau a son adresse dans le PAR 6 du noyau. Seuls 1024 Bytes de cette page sont utiles. C'est pourquoi le PDR 6 du noyau fixe les adresses permises dans la partie basse de la page et lui donne une grandeur de 1024 Bytes (Cfr fig. III.3).

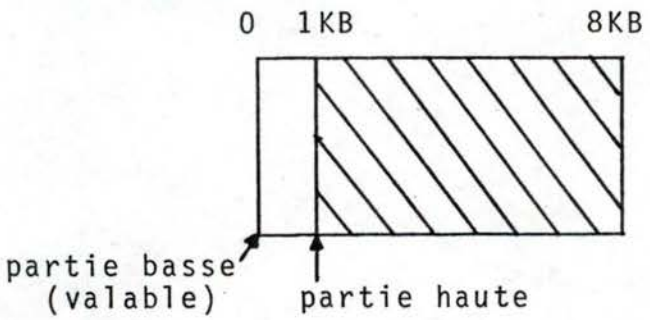


Fig. III.3 La page 6 de l'espace noyau

Pour ne pas trop entrer dans les détails, disons que le deuxième élément prend 16 KB (soit 2 pages virtuelles). Mais il est référencé en mode utilisateur. Ses adresses sont

donc obtenues par les PAR du mode utilisateur. Supposons que ce soient les PAR 2 et 3 du mode utilisateur. Pour ne pas perdre de la place, le PAR 2 de l'utilisateur pointe juste après la partie basse de la page 6 de l'espace noyau. La partie valable de la page 2 de l'espace utilisateur prend toute la page. La page 3 utilisateur commence juste après la page 2 utilisateur et a aussi sa partie valable étendue sur toute la page.

Finalement on obtient la description de la figure III.4 .

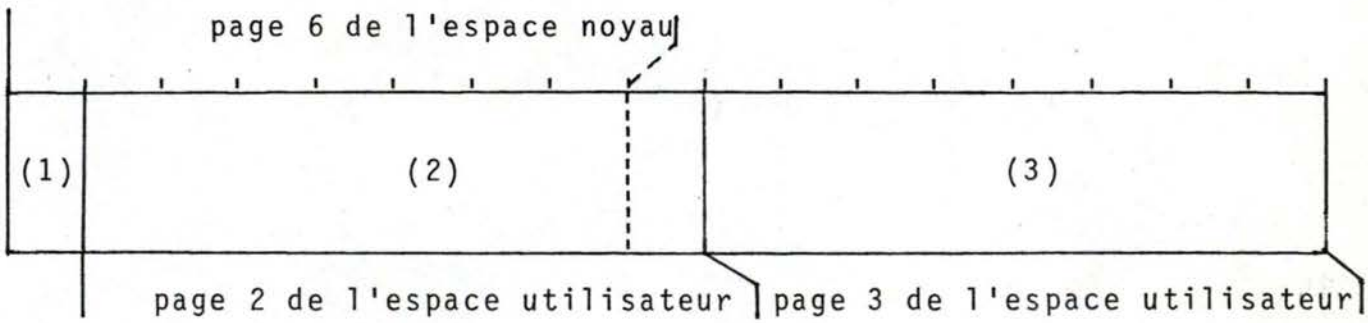


Fig. III.4 Description par page du segment de donnée

Légende :

(1) partie valable de la page 6 de l'espace noyau
(2) partie valable de la page 2 de l'espace utilisateur
(3) partie valable de la page 3 de l'espace utilisateur

Cet exemple montre comment cette propriété des pages virtuelles permet d'économiser de la mémoire.

Il montre aussi pourquoi le premier élément du segment ne peut s'accroître. En effet s'il le fait, il occupe de la place du deuxième et détruit donc de ses informations (cfr III.3.2.1 Description des composants d'une image d'un processus).

III.3.1.4 Allocation et désallocation.

Quand il n'y a plus d'espace en suffisance en mémoire principale pour charger un processus, un autre doit être déchargé de la mémoire principale vers la mémoire auxiliaire se trouvant sur le disque.

Cette mémoire auxiliaire sert purement à contenir temporairement des segments de processus.

L'allocation, ainsi que la désallocation, des deux mémoires est gérée par la même routine. Cependant l'unité mémoire accordée ou libérée est de 32 mots pour la principale et de 256 mots pour l'auxiliaire.

Pour chacune, une liste des zones utilisables est contenue dans un tableau de dimension fixe qui est égale àCMAPSIZ (mémoire principale) ou à SMAPSIZ (mémoire auxiliaire). La description d'un trou en mémoire est donnée par deux éléments du tableau; le premier donne la grandeur, le second l'adresse. Ce fait peut sembler étrange mais est dû à un raccourci dans les déclarations (cfr /18/ LIONS J., pour plus de détails).

Les listes sont classées par adresse, de la plus petite vers la plus grande. Quand un processus a besoin d'un trou, il consulte la liste et prend le premier qui lui convient (algorithme du "first-fit").

Lors de la désallocation, deux trous adjacents sont réunis en un seul.

III.3.2 Les processus.

Un processus peut être considéré comme une entité inactive sur laquelle on agit par des entités actives comme le processeur.

RITCHIE et THOMSON parlent d'image de processus. L'image est l'état courant d'un pseudo-ordinateur c'est-à-dire d'une structure de données abstraites qui peut être représentée dans une autre mémoire. Le processus n'est autre que l'exécution d'une image.

Dans UNIX, un processus peut créer ou détruire un autre; il peut acquérir et posséder des ressources. Son existence est impliquée par l'existence d'une structure non vide dans le tableau "Proc". Chaque processus possède une "Per Process Data Area" dans laquelle se trouve une copie de la structure "User".

L'image d'un processus comporte deux ou trois zones mémoires physiquement distinctes : la structure "Proc", le segment de données et le segment de texte .

III.3.2.1 Description des composants d'une image d'un processus.

A. La structure "Proc".

Elle est l'élément d'un tableau nommé aussi "Proc" qui a une dimension fixe égale à NPROC. Ce tableau réside en permanence dans la mémoire principale. Tout processus possède le long de sa vie un de ses éléments.

Un élément sert de descripteur de processus. Il donne des indications sur l'état du processus (endormi, prêt à tourner, ...), sur la fonction qui le manipule (chargement, ...), sur sa priorité; il contient aussi des pointeurs qui permettent de retrouver ses différents autres composants.

On ne peut avoir plus de NPROC processus en concurrence.

B. Segment de texte.

C'est une zone qui contient uniquement du texte pur et des données invariables. Elle ne fait l'objet d'aucune modification.

Elle n'est présente que pour des processus qui utilisent du code réentrant . Elle peut donc être partagée par plusieurs processus.

Quand un processus commence l'exécution d'un programme qui utilise du code réentrant , on regarde dans le tableau des segments de texte s'il existe déjà un processus utilisant ce programme. S'il n'y en a pas, on crée une copie du segment de texte dans la mémoire auxiliaire et une entrée inutilisée dans le tableau des segments de texte est réservée pour référencer la copie. S'il en existe déjà un qui utilise le même segment de texte, on lie le processus à ce même segment. Le tableau des segments de texte a une dimension fixée à NTEXT. Chaque élément donne des indications sur le segment de texte qu'il référence (adresse, longueur, ...).

C. Le segment de donnée (fig. III.5).

Il en existe un par processus dans le système. Il se compose de trois zones. Il est adressé en mode utilisateur.

- La pile utilisateur.

C'est une place réservée pour sauver les registres généraux r5 et r6 en mode utilisateur.

Ces registres sont liés au processeur et sont toujours accessibles; r5 est appelé le pointeur d'environnement et est utilisé pour donner la dernière activation de procédure enregistrée dans la pile courante; r6 ou le pointeur

de pile donne une zone de la pile où sont placées temporairement des variables.

Lorsqu'à un nouveau processus on donne un segment de donnée, on lui attribue d'office une place de SSIZE x 32 mots pour cette pile. Et s'il vient à en manquer par la suite, on l'agrandit de SINCR x 32 mots.

- La zone de donnée.

Elle contient du code non-réentrant, des données variables au cours du temps, etc...

- La "Per Process Data Area" (PPDA).

C'est une zone de l'espace noyau virtuel. Elle correspond toujours aux1024 premiers Bytes de la 7ème page virtuelle de cet espace.

Les Bytes 300 à 1024 sont réservées à la pile noyau qui sert à sauver les registres r5 et r6 en mode noyau. Cette place de 724 Bytes ne peut être agrandie (cfr III.3.1.3 La page). On croise les doigts pour que UNIX ne demande jamais plus d'espace que 724 Bytes pour la pile noyau !!!

Les 300 premiers Bytes sont destinés à contenir une copie de la structure "User" qui réside en permanence en mémoire centrale.

D. La structure "User".

L'original de cette structure ne sert qu'à être copiée dans la PPDA du nouveau processus.

Elle contient des informations utiles lorsque le processus est courant, des zones de sauvetages lors d'un déchargement, des données sur les entrées / sorties (E/S), les segments, les fichiers ouverts par le processus, des indications pour les registres de segmentation, etc...

Elle possède aussi un pointeur vers le descripteur du processus correspondant.

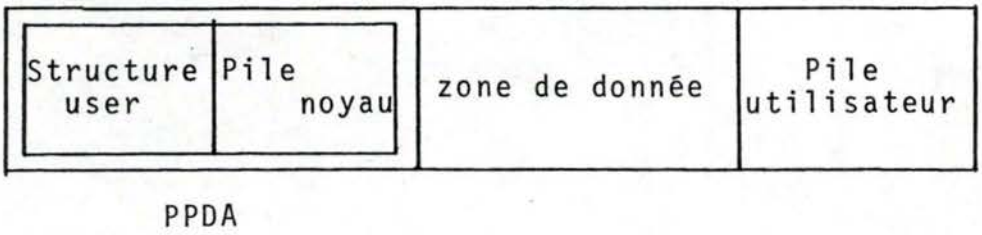


Fig. III.5 Le segment de donnée

III.3.2.2 Représentation des éléments d'un processus.

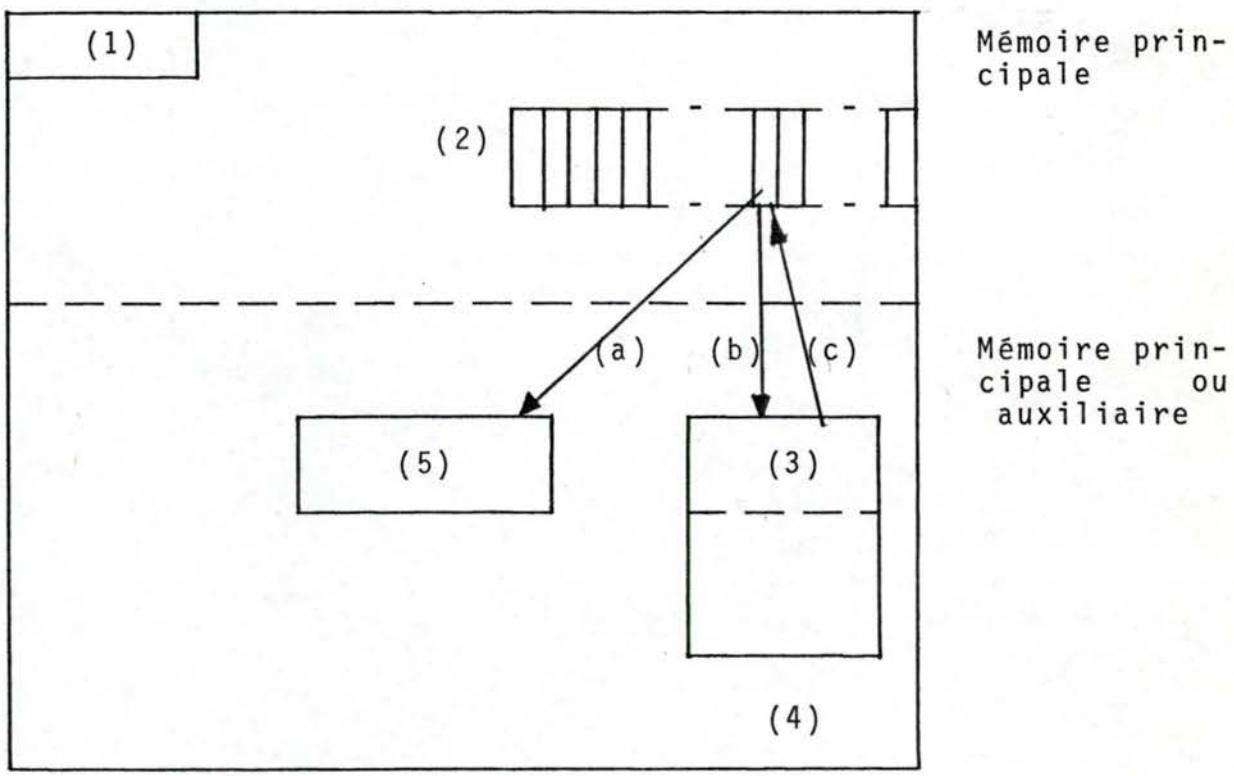


Fig. III.6 Composants de l'image d'un processus

(1) Original de la structure "User"
(2) Tableau des structures "Proc" servant de descripteur pour les processus
(3) "Per Process Data Area" d'un certain processus. Elle contient une copie de la structure "user"

(4) Le "segment de donnée" qui se compose de la "Per Process Data Area" et d'une autre zone pour des données et des piles
(5) Le segment de texte (éventuel)

(a) Pointeur de la structure "Proc" vers le segment de texte du processus
(b) Pointeur de la structure "Proc" vers le segment de donnée du processus
(c) Pointeur de la structure "User" de la "Per Process Data Area" vers la structure "Proc" correspondante

III.3.2.3 Exécution d'une image.

Un processus peut tourner en mode noyau quand le code UNIX tourne en son propre nom ou en mode utilisateur sinon. Par facilité on parle de processus noyau et de processus utilisateur.

Quand le gérant des processus en choisit un pour le processeur, il prend l'adresse du segment de donnée (dans le descripteur du processus) et positionne les registres de segmentation 6 du noyau pour qu'il pointe vers le PPDA du processus.

Chaque activation d'un processus utilisateur est précédée et suivie par une activation du processus noyau correspondant.

- Exécution en mode noyau.

Les registres de segmentation 6 du noyau sont destinés à adresser seulement le PPDA. Or celle-ci est constante et vaut 512 mots.

Les adresses virtuelles que le code UNIX peut accéder sans causer d'interruption internes sont :

0 – 48 KB (page 0 à 5 de l'espace noyau virtuel)
48KB – 48KB + 1024 Bytes (les 1024 premiers Bytes de la page 6 du même espace)
56KB – 64KB(page 7 du même espace).

Les valeurs de r5 et r6 restent dans les Bytes 300 à 1024 de la PPDA.

- Exécution en mode "utilisateur".

Quand un processus est en mode noyau, les registres de segmentation du mode "User" sont généralement bien positionnés pour ce processus. Cependant il ne réalisent la translation des adresses virtuelles en adresses physiques que pour l'espace utilisateur (c'est-à-dire en mode utilisateur).

Les registres des deux modes sont proprement positionnés chaque fois qu'un processus est en mode utilisateur.

Les registres r5 et r6 pointent dans la pile utilisateur qui en cas de débordement peut s'agrandir (vers le bas de la page 7 de l'espace utilisateur virtuel).

III.3.2.4 Gestion de processus (généralités).

A. Commutation de processus.

Une commutation se produit lorsqu'un processus actif se suspend lui-même et laisse le processeur disponible pour un autre processus.

Il existe quatre moments où peuvent se faire ces commutations :

- quand un processus se bloque (E/S, attente d'une ressource, ...),
- quand un processus noyau est sur le point de revenir en mode utilisateur (exemple : après avoir exécuté un appel système ou une interruption),
- quand un processus se termine,
- quand un processus demande un accroissement dans son allocation en mémoire principale, qui ne peut être accordée immédiatement.

B. Interruption.

Une interruption hardware ne provoque pas directement une commutation. Quand un processus utilisateur tourne, le processeur se branche lui-même automatiquement en mode noyau pour trouver quelle routine de service d'interruption doit s'exécuter. UNIX assure que de telles routines s'exécutent en mode noyau et tournent avec la plus haute priorité quel que soit le processus qui tournait lors de l'interruption.

Avant de se terminer, la routine regarde si on continue le processus interrompu ou si on commute.

Si un processus noyau est interrompu, alors, après que l'interruption se soit terminée, le processus noyau est toujours repris au point où il fut arrêté sans se soucier de savoir si un autre processus exécutable n'a pas une priorité supérieure. Ensuite ou bien on revient en mode"user"pour exécuter le processus utilisateur interrompu, ou bien on passe à l'exécution d'un autre processus en mode noyau.

III.3.3 Les tampons pour supports de type bloc.

Il existe des supports de type bloc et de type caractère. Les premiers sont appelés ainsi parce qu'ils sont adressables par bloc de 512 Bytes. Les seconds envoient et reçoivent par caractère.

Un tampon effectif pour bloc comprend un descripteur et une zone mémoire de 514 caractères.

Le descripteur se représente par une structure "buf". Elle est l'élément d'un tableau appelé aussi "buf", qui a la dimension fixe déterminée par le paramètre NBUF.

Elle contient des indicateurs sur l'état du tampon (transmission réussie, lire quand E/S se produit, ...), des pointeurs vers deux listes ("b_list" et "av_list"), des paramètres pour l'actuel transfert de données, un pointeur vers une zone.

Cependant si à toute zone correspond un descripteur, à tout descripteur ne correspond pas nécessairement une zone de 514 caractères car un descripteur peut être utilisé purement comme place pour des arguments lors d'exécutions de routines d'E/S afin de les décrire.

Les descripteurs peuvent être liés simultanément à deux listes.

Il existe une "b_list" par support de type bloc, qui peut être vide éventuellement.

L' "av_list" comporte des tampons qui peuvent être détachés de leur emploi courant et convertis à un double emploi. Ainsi l'appel système "exec" fait appel à un tampon pour y placer temporairement des arguments.

L'appartenance d'un tampon à une liste ou à une autre est donnée par des indicateurs dans le descripteur.

Les tampons ne sont accordés que pour de courts intervalles à un processus.

L'information est laissée dans le tampon tant qu'il est nécessaire.

C'est le système qui décide lorsqu'un tampon doit se vider, ce qui pour certains peut prendre un certain temps. C'est pourquoi un programme utilitaire oblige tous les tampons à se vider inconditionnellement deux fois par minute.

C'est ce genre de tampon qui est utilisé pour le chargement et le déchargement de processus en mémoire principale.

III.3.4 Les fichiers.

III.4.1 Système de fichier.

Au point de vue logique, un fichier se trouve dans un système de fichiers. Celui-ci est une collection intégrée de fichiers avec un système hiérarchique de directoires enregistré sur un simple bloc pour support de stockage. Un support d'accès est un mécanisme pour transférer de ou vers un support de stockage.

Au point de vue physique, un fichier se trouve sur un support de stockage. Celui-ci a pour définition de pouvoir stocker de l'information. Il est acceptable comme volume de système de fichier si trois conditions sont respectées.

D'abord l'information est enregistrée sur le support en blocs adressables de 512 Bytes et qui indépendamment peuvent être lus ou écrits.

Ensuite le support doit obéir à quelques critères d'organisation. Le bloc 1 doit être formé comme un super bloc. Les blocs 2 à (n+1) (où n est contenu dans le super bloc) contiennent une table "Inode" qui référence tous les fichiers enregistrés sur le support et ne référence aucun autre fichier.

Enfin les fichiers de directoires enregistrés sur le support référencent uniquement tous les fichiers qui sont sur ce support.

Le super bloc contient des informations utilisées dans l'allocation de ressources : des adresses de blocs utilisables, des entrées dans la table "Inode" du volume.

Un support de stockage est accessible si il est inséré dans un support d'accès. Un volume de système de fichier est monté, c'est-à-dire est utilisable, si la présence du support de stockage dans un support d'accès a été formellement reconnue par le système d'exploitation.

Tant que le volume est monté, une copie du super bloc se trouve dans un tampon de type bloc et y est mis à jour. Cette copie est transmise à intervalle régulier au support de stockage pour éviter des ennuis (cfr Les tampons pour supports de type bloc).

Un volume monté possède une entrée dans le tableau "Mount". Celui-ci est de dimension fixe déterminée par TMOUNT. Un élément comporte un pointeur vers le tampon qui contient le super bloc du volume, un pointeur vers sa table "Inode", des indications sur le support où se trouve le volume.

III.3.4.2 Accès aux fichiers.

Tout fichier référencé par un processus a une entrée dans le tableau "File" de dimension fixe donnée par le paramètre NFILE. Un élément donne le nombre courant de processus qui l'utilise, des indications sur son usage (lecture, écriture, ...), un pointeur vers le tableau "Inode" de la mémoire principale.

Un processus, par sa structure "User", possède un pointeur vers chacun des éléments du tableau "file", qu'il s'est vu accordé. Pour que cette règle puisse s'appliquer à tout processus, il faut que NFILE soit suffisamment grand, c'est-à-dire

NFILE > NOFILE x NPROC

où NPROC est le nombre maximum de processus en concurrence.

En fait cette relation n'est pas tout à fait juste car les processus "enfants" héritent des pointeurs vers les éléments du tableau "File" du processus "parent" (cfr III.2.1 Les primitives). RITCHIE et THOMSON proposent NOFILE = 15, NPROC = 50 et NFILE = 100.

III.3.4.3 Le Tableau "Inode".

Il ne faut pas le confondre avec la table "Inode" qui se trouve dans tout volume de système de fichier. Ce tableau réside dans la mémoire principale en permanence. Il est de dimension fixe donnée par NINODE.

En tout temps, il existe une et une seule entrée pour chaque fichier qui peut être référencé pour des opérations, qui est exécuté, qui a été exécuté et sera prochainement réutilisé ou qui est le directoire de travail pour un processus. Plusieurs entrées du tableau "File" peuvent pointer vers une entrée dans le tableau "Inode" mais jamais deux entrées de celui-ci ne correspondont à une entrée du premier.

Un élément donne des informations sur le dernier bloc référencé du fichier, sur son mode, sur son directoire, sur son support,...

III.3.4.4 Directoires de fichier et fichiers de directoire.

L'information relative à un fichier individuel se trouve dans les tables "Inodes" elles mêmes sur disque ou/et en mémoire centrale (dans le tableau "Inode").

Les noms des fichiers ne se trouvent pas dans les tables "Inodes". Ils sont stockés dans les fichiers de directoire . Chaque nom définit un et un seul fichier. Un fichier possède un ou plusieurs noms. Ceux-ci ont la forme a/b/c; a/x et a/1 sont 2 fichiers reliés (ils peuvent appartenir au même utilisateur).

Les utilisateurs font référence initialement au fichier en donnant le nom du fichier. Un fonction système se charge de décoder le nom dans l'entrée "Inode" correspondante. Pour ce, UNIX crée et maintient une structure de données de directoire : un graphe dirigé avec des noeuds nommés. Dans la

forme la plus pure c'est un arbre. Plus généralement dans UNIX, le graphe est un treillis qui peut être obtenu d'un arbre en fusionnant un ou plusieurs groupes de feuilles. Dans ce cas, bien qu'il y ait toujours un seul chemin entre un noeud intérieur et la racine, il peut y avoir plus d'un chemin entre la racine et une feuille.

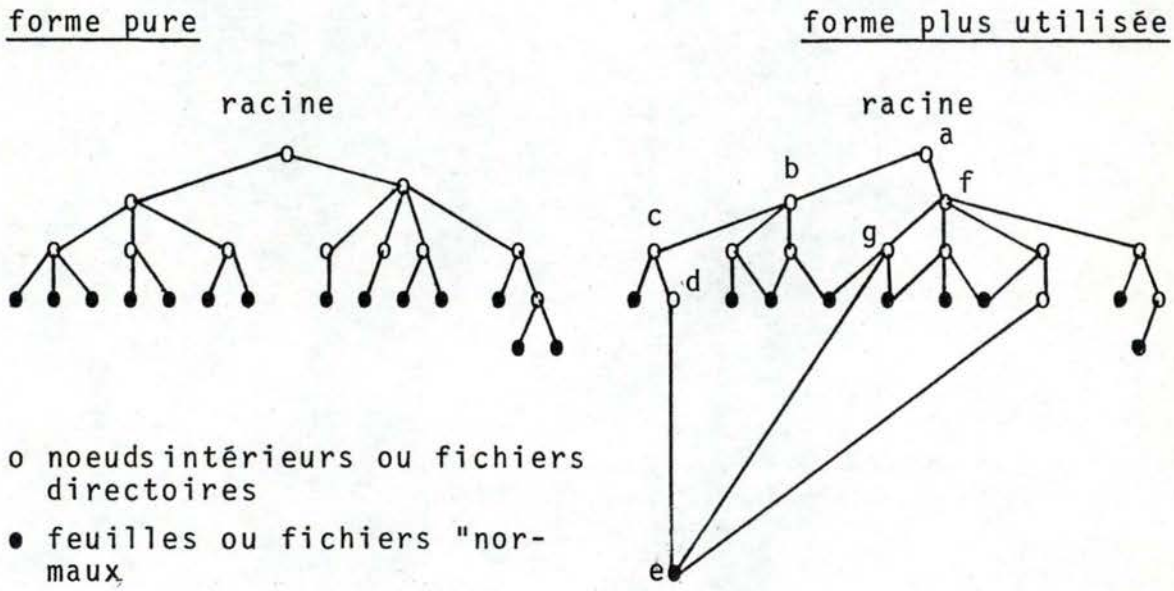


Fig. III.7 Structure des noms de fichiers

Un fichier de directoire et un fichier qui ne contient que des noms de fichiers. Le nom d'un fichier est la succession des noms des noeuds rencontrés le long du chemin allant de la racine au fichier en question. S'il y a plusieurs chemins, il existe plusieurs noms.

Exemple : a/b/c/d/e ≡ a/f/g/e

Ces fichiers de directoire contiennent de l'information nécessaire à la localisation d'autres fichiers, ce qui implique une grande protection et leur gestion est laissée uniquement au système d'exploitation.

Dans tout fichier l'information est stockée en un ou plusieurs blocs de 512 caractères. Chaque bloc d'un fichier de directoire est divisé en 32 éléments. Chaque élément représente un nom de fichier. Il contient un pointeur vers une entrée de la table "Inode" du volume et le nom d'un noeud.

III.3.4.5 Composants d'un fichier.

En guise de récapitulation, un fichier existant mais non référencé occupe une entrée dans un fichier de directoires, une entrée dans la table "Inode" sur le support, zéro, un ou plusieurs blocs de stockage sur le support.

S'il est référencé pour une raison ou une autre, il possède une entrée dans le tableau "Inode" de la mémoire principale.

Enfin si un processus a ouvert un fichier pour y lire ou y écrire, il occupe une entrée dans le tableau "File" et un pointeur vers celle-ci dans la structure "User" de la PPDA du processus.

III.4 La configuration.

III.4.1 La configuration minimale.

Le minimum de matériel nécessaire à l'utilisation de UNIX se compose de :

- un processuer PDP 11/34, 11/40, 11/45, 11/60 ou 11/70;
- option "Memory Management";
- 48 Kmots de mémoire centrale;
- 1 horloge temps réel;
- 1 disque de 5 MBytes;
- 1 terminal/console.

III.4.2 La configuration utilisée.

Nous avons pu utiliser la configuration suivante :

- processeur DEC-PDP 11/45 avec
  - option "Memory Management",
  - option "Floating Point Processor",
  - mémoire centrale de 128 KBytes,
  - 1 disque DEC-RP05 de 100 MBytes,
  - 1 dérouleur de bande DEC-TU16 (800/1600 bpi),
  - 1 imprimante DEC-LA 180 (180 cps),
  - 1 console DEC-LA 36,
  - 1 multiplexeur 16 lignes DEC-DH11.
- processeur DEC-PDP 11/70 avec
  - option "Memory Management",
  - option "Floating Point Processor",
  - mémoire centrale de 128 KBytes,
  - 1 disque DEC-RP05 de 100 MBytes,
  - 2 dérouleurs de bandes compatibles (800/1600 bpi),
  - 1 imprimante rapide compatible (600 lpm),
  - 1 multiplexeur 16 lignes DEC-DH11,
- 16 terminaux VDU, de type DEC-VT52 et compatibles.

Etant donné notre problème, les mesures de performances ont été effectuées sur l'ordinateur DEC-PDP 11/45, alors que la mise au point des programmes s'est faite sur le PDP 11/70.

Chapitre IV

THEORIE DE LA CHARGE

Jean VERCHEVAL

Ce chapitre est destiné à décrire brièvement quelques notions théoriques liées à la charge principalement dans le cadre des mesures de performance sur S3 (cfr I.5 - Définition de performances de S3, p.23). Il aborde certains problèmes qui se posent, avec, parfois, l'ébauche d'une solution.
Il souligne aussi les multiples facettes que peut montrer la construction d'une charge.

IV.1 Définitions.

Définir une charge n'est pas aussi simple qu'il y paraît à priori. On peut considérer la charge d'un système informatique comme étant le travail qui s'y exécute. Mais affirmer que la charge est un ensemble cohérent de courants de transactions appliquées aux différentes ressources d'un système, est tout aussi valable.

Bien qu'apparemment semblables, ces deux définitions se distinguent par le point de vue adopté. La première est plus orientée observateur extérieur qui voit entrer les travaux dans le système. La seconde est manifestement rapprochée d'un parti interne qui voit une succession de transactions qui passent d'un service à l'autre du système.

IV.2 La charge et les mesures.

Un même système avec des charges aux caractéristiques différentes, donne à un même test de mesures des résultats différents et difficilement comparables. Il suffit pour s'en convaincre de prendre les cas d'une charge

purement composée de programmes de calcul et d'une autre ne contenant que des programmes réalisant beaucoup d'entrées-sorties. Quelle que soit la mesure effectuée, taux d'utilisation, temps de réponse, les résultats au test ne concordent pas. La dépendance entre la charge et les mesures est très forte. Et si, pour une raison ou l'autre, des mesures sont faites sur une charge non représentative de la réalité sur laquelle on travaille, il ne reste plus à l'expérimentateur qu'à tout recommencer ou à n'exposer qu'un modèle théorique imaginaire.

Afin d'éviter une telle impasse, il existe trois façons de procéder :

- On mesure directement sur la charge réelle. Mais dans ce cas un nombre assez important de tests doit être fait afin que les résultats représentent bien un échantillon statistiquement valable. Une telle mesure s'étale souvent sur une très longue période.
- A partir d'une charge réelle, on construit un modèle sur lequel on teste. Des mesures sur la charge sont pratiquées pour l'analyser. Cette étude donne un modèle que l'on teste afin de s'assurer qu'il correspond bien à la charge réelle. Si ce n'est le cas, il faut recommencer. Sinon, les mesures souhaitées sont entreprises dès lors sur le modèle.
- La charge réelle n'existe pas encore. Dans ce cas, il faut établir des critères sur la charge future, en essayant de la comparer à des charges réelles déjà existantes, en analysant l'emploi futur du système. L'étude de ces données débouche sur la formation d'un modèle théorique de la future charge, sur lequel on teste.

IV.3 Représentation d'une charge.

Des deux définitions de la charge découlent deux façons possible d'aborder sa représentation. Soit on essaie d'établir des programmes dont le but est de simuler le taux d'occupation des différentes ressources du système, soit on tente de construire des programmes typiques de l'utilisation de l'ordinateur.

Lorsqu'on ne dispose pas d'une charge réelle, il est difficile, voire impossible d'adopter la première solution.

Dans le deuxième cas, il faut d'abord collecter toute une série d'informations pendant un certain temps; relever les différents travaux demandés et leur date de lancement, la manière dont ils sont introduits, leur fréquence, les fluctuations éventuelles des temps de réponse, ... Ensuite, l'étude de ces données permet de rassembler les travaux par type (édition, compilation, ...) avec leur fréquence d'application; elle fournit un scénario dans leur succession à l'exécution; elle donne une liste des singularités distinguées. Le fait de ne pas disposer d'une charge réelle dans ce cas, n'est pas trop gênant. En effet, définir une charge future est avant tout fournir les indications énoncées ci-avant.

Une fois cette étude terminée, on écrit les programmes du modèle. Ceux-ci peuvent être classés en deux groupes :

- Les programmes synthétiques qui représentent les types de travaux.
- Les programmes singuliers qui exécutent certaines particularités qui jouent un rôle assez important dans la charge réelle (ou future).

Il reste à déterminer leur ordre de chargement et la manière de les lancer pour avoir terminé la construction d'une charge représentative.

IV.4 Reproduction d'une charge représentative.

Un modèle clair et précis peut être obtenu sans qu'il soit possible de le mettre intégralement en pratique.

Lors de l'emploi de l'interactif, il se peut qu'un utilisateur se trompe fréquemment dans la frappe des caractères, qu'un autre marque de longs temps de réflexion, qu'un autre encore écrive et lance plus vite les commandes. Il se peut que l'importance de ces facteurs amène l'analyste à en tenir compte dans son modèle. A moins de le faire manuellement, il est difficile de simuler parfaitement le rôle de tels facteurs (cfr IV.5 Construction d'une charge interactive, représentative et reproductible).

De même il se peut que le modèle soit tellement important en travaux que le temps réservé aux tests ne suffise pas pour fournir des conclusions valables.

Un modèle qui fait appel à des fonctions inexistantes sur le système testé, est tout à fait inutile.

IV.5 Construction d'une charge interactive, représentative et reproductible.

On suppose que la construction de la charge représentative a fourni n scenarii pour les n terminaux de la configuration à tester. Plusieurs approches sont possibles pour le reproduire.

- Avec personnel.

On demande à n opérateurs de s'installer devant un terminal avec un scénario à exécuter.
C'est cher et dangereux car l'erreur est humaine.

- Avec des terminaux à cassette.

On remplit une cassette d'un scénario et on la lance. Mais ainsi les commandes sont crachées à la suite des autres; il faut pouvoir employer des "stops" avec des "starts". Mais on ne tient pas compte des temps de réflexion et le débit de frappe est plus rapide qu'avec un opérateur.

C'est une solution viable mais il reste le problème du débit. Peut être qu'en diminuant le nombre de terminaux, il soit possible de le contrer !

- Avec "BIBO" (Batch in batch out).

On simule à l'intérieur du système central les n terminaux avec leurs messages.

On biaise le comportement des lignes mais on dispose d'une certaine souplesse au niveau du débit.

- Avec un deuxième ordinateur.

Il simule aisément les n terminaux avec leur scénarii. Il dispose d'une source de synchronisation pour régler les problèmes de débit. Il peut même servir d'outil de mesure.

C'est idéal mais cher, complexe et encombrant.[1]

---

1. Pour plus de détails, cfr P. De RIVET et C.A. ROSE.

Chapitre V

CONSTRUCTION ET LANCEMENT DE LA CHARGE

Jean VERCHEVAL

Ce chapitre donne d'abord un aperçu des façons d'aborder les mesures de performance et le choix effectué pour notre mémoire.

Ensuite, il montre la construction du modèle de charge de S3 et les problèmes qui poussent à le réviser.

Il décrit aussi quels types de mesure sont faits.

Il fournit des indications sur l' "overhead" et la manière dont certains de ses éléments sont considérés.

V.1 Construction de la charge.

V.1.1 La charge d'une université.

L'utilisation d'un système informatique dans une université soulève plusieurs problèmes. Essentiellement, il s'agit de la grande diversité tant dans les catégories d'utilisateurs, que dans le type des travaux.

En ce qui concerne les types de travaux, on peut distinguer :

1. Travaux importants de recherche, utilisant beaucoup de ressources.
2. Masse importante de traitements par lots, constituant les travaux pratiques des étudiants pour les cours d'informatique.
3. Grande utilisation interactive faisant également intervenir des quantités élevées de ressources.
4. Contrôle d'expériences en temps réel.
5. Manipulation de bases de données.

Un autre type de diversité existe dans la population des utilisateurs, et plus particulièrement dans leur compétence, leur savoir-faire à l'égard de l'ordinateur.

Cela implique une relative souplesse du système d'exploitation dans la gestion des demandes résultant des différents types d'utilisation des ressources et des différents types de travaux.

Une autre difficulté dans la détermination de la charge provient de la distribution dans le temps des demandes. En effet, la charge est caractérisée par des fluctuations dépendant de la période de l'année (par exemple, certains travaux pratiques ne seront organisés que certains semestres seulement).

Ici encore, le système informatique doit pouvoir faire face et s'adapter à la variation de la charge.

Afin d'étudier le comportement du système, il est nécessaire de diviser l'utilisation en les classes énoncées ci-avant.

Cette classification permettra de définir avec exactitude le type de charge auquel une certaine machine sera soumise.

Dans le cas d'un ordinateur unique, il s'agit de déterminer la combinaison des différents types de charges, et éventuellement la variation temporelle (qui peut être difficile à prévoir).

Dans le cas de plusieurs machines, la tâche de l'organisateur peut être simplifiée s'il est possible d'attribuer une certaine charge spécifique à un ordinateur. Même alors, la variation temporelle reste un élément qui risque d'être incontrôlable.

V.1.2 La charge de S3.

Le système informatique S3 des Facultés ne nous a pas été mis à disposition car il n'est pas encore installé. Du reste, sa configuration réelle ne nous est pas connue. Nous savons qu'il s'agit sûrement d'un ordinateur DEC PDP 11/45.

De plus, le type de travaux auquel il sera consacré n'est pas encore clairement défini.

Les seuls éléments sûrs pour construire la charge sont donc réduits à quelques caractéristiques fort générales :

- la charge doit comporter un ensemble de petits travaux assez variés;
- ceux-ci correspondent à des applications d'étudiants ou de chercheurs;
- elle doit pouvoir s'adapter à un moniteur de test donnant des temps de réponse (cfr I.2.4 - Choix de l'objet de mesures);
- il n'y a pas de charge réelle.

Plutôt que d'élaborer directement une charge à partir d'indices aussi vagues, la préférence est donnée à une étude basée sur un modèle de charge construit pour une université. Or les "benchmarks" utilisés pour le choix du nouveau système informatique des Facultés N-D de la Paix, dont le nom de code est S12, rentre bien dans le cadre de cette aspiration.

Le benchmark B10 est considéré comme le test d'acceptation de S12. Il doit donc contenir les éléments principaux d'une charge universitaire. C'est pourquoi nous le prenons comme point de départ.

Des critiques sur ce choix peuvent être aisément trouvées : S3 n'est pas S12, ils n'ont pas le même usage, ... Mais faute de renseignements sur S3, il est à peu près impossible d'arriver à des conclusions nettes et précises sur sa charge future.

V.1.3 UNIX et la charge.

La construction de la charge doit tenir compte de la reproduction de celle-ci (cfr IV.4 - Reproductibilité de la charge). Ainsi la version standard de UNIX dont nous disposons, implique le respect d'un certain nombre de principes.

- Le modèle de charge ne doit pas comporter de programmes Algol, Cobol ou APL. UNIX ne possède pas de compilateurs pour ces langages.
- UNIX dispose d'un dialecte BASIC assez limité et peu utilisable pour des applications de quelque importance, ce qui implique un emploi modéré du BASIC présenté par UNIX.
- Il possède aussi un compilateur FORTRAN et un compilateur pour le langage C . L'emploi de ce dernier sur UNIX est tellement répandu qu'il occupe une place importante dans l'utilisation des différents langages.

Il faut ajouter qu'intrinsèquement, UNIX est un système purement orienté interactif et n'est pas conçu pour l'exécution de très gros programmes de calcul (la tendance semble être de pénaliser fortement de tels programmes au point d'en décourager impitoyablement des utilisateurs éventuels).

V.1.4 Description de B10.

(tiré de /10/ FacultésUniversitairesde Namur).

But du benchmark B10.

Il s'agit de tester si le niveau de performance exigé sera atteint, c'est-à-dire :

- exécuter N_B travaux batchs courts (<30 sec. CPU siemens 4004/151) en une heure, avec un temps de transit moyen du job $T_T \leqslant 30$ min;

- simultanément, servir N_S sessions interactives, consacrées à de l'édition ou de la consultation de fichiers, avec un Temps de réponse T_R moyen

$$\overline{T_R} \leqslant 2 \text{ sec} \quad \text{si } N_S \leqslant 0.66\ N_P$$

$$\overline{T_R} \leqslant 5 \text{ sec} \quad \text{si } N_S = N_P$$

où N_P est le nombre de portes réservées aux terminaux interactifs;

- simultanément, assurer le service des N_R lignes remote-batc;
- simultanément, assurer les autres travaux à montage.

N_B : nombre de travaux batch inférieurs à 30 sec par h

N_T : nombre de terminaux alphanumériques ou graphiques installés

N_P : nombre de portes réservées aux terminaux alphanumériques ou graphiques

N_R : nombre de portes réservées aux terminaux lourds.

	1979	1981
N_B	10	12,5
N_T	35	46,0
N_P	25	32,0
N_R	7	7,0

Composition du benchmark.

A. Travaux batch.

8 exécutions (PR1, PR2, PR4, PR5, PR6, PR7, PR8, PR9).
2 compilations (compilation de PR3 et PR10) interviennent dans le benchmark B10.
PR3 et PR10 sont compilés pendant l'exécution de B10.

PR7 est un programme batch effectuant de nombreuses entrées/sorties.

Le temps de transit de chaque batch doit être inférieur ou égal à 30 minutes.

B. Interactif.

La charge stimulant l'interactif consiste en une série de travaux interactifs décrits dans les séries 1 à 7.

Charge interactive pour 1979.

Le nombre de portes (c'est-à-dire nombre maximum de terminaux actifs simultanément) est de 25 (N_P=25).

Il faut que le temps de réponse (des commandes éditeurs ou de consultations de fichiers) soit :

$\overline{T_R} \leqslant 2$ sec si $N_S \leqslant 0.66\ N_P$

$\overline{T_R} \leqslant 5$ sec si $N_S = N_P$

le Benchmark devra donc être effectué dans deux environnements différents :

- l'un avec 17 terminaux (N_S = 17) : test 1
- l'autre avec 25 terminaux (N_S = 25) : test 2

Deux types de terminaux devront être connectés :

- vidéos (75%);
- téléimprimeurs (25%).

Profil de la charge interactive.

	nombre de terminaux (test 1)	nombre de terminaux (test 2)
COBOL	3	4
FORTRAN	9	12
BASIC	1	2
ALGOL	1	2
EDITEUR DE TEXTE	3	5
	17	25

numéro de série	terminaux concernés (test 1)	terminaux concernés (test 2)
1	T1, T2, T3	T1, T2, T3, T18
2	T4, T5, T6	T4, T5, T6, T19
3	T7, T8, T9	T7, T8, T9, T20
4	T10	T10,T21
5	T11	T11,T22
6	T12, T13, T14	T12,T13,T14,T23
7	T15, T16, T17	T15,T16,T17,T24,T25
Total bombre de sessions (N_S)	17 dont 4 téléimprimeurs	25 dont 6 téléimprimeurs

sur téléimprimeur (300 BPS)

les autres : vidéos(2400 BPS)

C. Remote-Batch.

Le nombre de Remote-Batch actifs simultanément est de 7. Toutes les 10 minutes, chacun d'eux enverra au système central le programme PR8 (perforé préalablement sur cartes). L'ordinateur renverra 2 fois le fichier contenant ce programme à l'imprimante locale toutes les 10 minutes.

MESURES :

Pour chaque Remote-Barch et envoi de cartes, on demande le temps entre le moment où la dernière carte est lue et celui où la première ligne est imprimée.

VITESSE DE LIGNE :

600 caractères par seconde (4800 bps).

D. Travaux de montage.

Afin de voir l'influence des travaux de montage, il s'agit d'exécuter pendant une heure le programme de manipulation de bande suivant : (appelé PR11)

```
      DIMENSION IBUF (1000)
      DO 10 I = 1,1000,1
   10 IBUF(I) = I
   30 DO 20 J = 1,1000,1
   20 WRITE (1) IBUF
      REWIND 1
      GOTO 30
      STOP
      END
```

Nous donner le nombre de REWIND effectués sur une heure.

Pour plus de détails : cfr /10/ Facultés universitaires de Namur.

Description détaillée des sessions interactives.

Les sessions sont des développements de programme ou de l'édition de fichiers et comportent une série de tâches à exécuter au terminal.

Chaque étape est accompagnée d'un commentaire définissant des temps, etc...

Chaque session est à répéter sur le terminal correspondant autant de fois qu'il est nécessaire pendant une heure.

Les sessions identiques sont groupées en série.

V.1.5 Transformation de la charge de B10.

Les paragraphes précédents montrent que notre charge ne peut être tirée directement du benchmark B10. Il faut lui apporter certaines modifications.

L'élimination du "batch" et du "remote-batch" est nécessaire car ils ne correspondent pas à la philosophie de UNIX.

La partie "travaux de montage" est aussi supprimée car elle sera sûrement peu importante dans l'emploi futur de S3.

Pour compenser un peu cette perte, le programme "batch" PR7 est placé dans la partie interactive. Il est choisi parce qu'il effectue beaucoup d'entrées/sorties, ce qui rentre assez bien dans le schéma global d'utilisation de UNIX.

Les programmes de la partie interactive écrits en Algol et en Cobol sont traduits dans le langage C car ils ne peuvent tourner tels quels sur UNIX (cfr V.1.3. UNIX et la charge) et des programmes doivent être écrits pour représenter l'emploi du C .

Le programme PR7 est aussi traduit en C pour renforcer l'importance de ce langage.

Nous obtenons ainsi une charge (C10M) qui est comparée à la charge initiale (C10) du test 1 dans le tableau V 1.

Cependant C10M est une charge qui se répartit sur 20 postes de travail. Il est presque certain que S3 n'en supporte pas un tel nombre. De plus, il n'est prévu aucune téléimprimante. C'est pourquoi nous postulons six vidéos actifs connectés simultanément à S3.

Mais une telle hypothèse entraîne une restructuration du profil de la charge.

Si dans B10, se trouvent les différentes séries évoquées, c'est qu'elles sont représentatives de travaux typiques qui peuvent être demandés au système, c'est-à-dire en fait de la charge réelle que l'ordinateur aura à supporter. Le nombre de terminaux attribués à chaque série peut-être considéré comme représentatif de la quantité de chaque travail typique par rapport aux autres.

Si l'on veut garder cette représentation mais en imposant la restriction qu'il ne peut y avoir qu'un seul terminal, il faut qu'à partir de celui-ci, soit exécutéela liste suivante de travaux dont l'ordre importe peu :

3 fois la série 1
3 " " " 2
3 " " " 3
1 " " " 4
1 " " " 5
3 " " " 6
3 " " " 7
3 " " " 8

Evidemment de cette manière, il n'existe plus de concurrence entre les différentes sessions mais nous gagnons une meilleure représentation de ce qui peut être demandé dans le temps à partir d'un terminal.

C 10				→	C 10 M			
Partie interactive				→	Partie interactive			
série	programme	langage	Nbre de terminaux		série	programme	langage	Nbre de terminaux
1	PI1	fortran	3/4	→	afb 1	afpi 1 f	fortran	3
2	PI2	fortran	3/4	→	afb 2	afpi 2 f	fortran	3
3	PI3	fortran	3/4	→	afb 3	afpi 3 f	fortran	3
4	PI4	basic	1/2	→	afb 4	bas 4	basic	1
5	PI5	algol	1/2	→	afb 5	afpi 5 c	c	1
6	PI6	cobol	3/4	→	afb 6	afpi 6 c	c	3
7	Editeur	-	3/5	→	afb 7	édit 7	shell	3
Partie batch				→	Interactif			
programmes		langage			série	programme	langage	Nbre de terminaux
PR1		-		→				
PR2		-		→				
PR3		-		→				
PR4		-		→				
PR5		-		→				
PR6		-		→				
PR7		fortran		→	afb 8	afpi 8 c	c	3
PR8		-		→				
PR9		-		→				
PR10		-		→				
Remote batch				→	——			
-				→				
Partie Montage				→	——			
				→				

Tableau 4 : Comparaison entre les charges C10 et C10M.

Cependant, nous recréons une concurrence en prenant un certain nombre de postes qui ont la même tâche que le premier à exécuter. Les principes sur la représentativité de la charge sont toujours respectés.

Nous parvenons ainsi à nous ramener à six vidéos mais nous perdons le bénéfice du choix de la concurrence entre les différents postes pour une autre moins définissable dans le temps.

Nous obtenons une charge qui reste représentative du contenu de la partie interactive de B10 et de la fréquence d'emploi de ses constituants. Elle est essentiellement composée de programmes synthétiques (cfr IV.3 Représentation d'une charge).

Nous l'avons baptisée CT (charge des tests).

V.1.6 Description de la charge CT (cfr Annexe A5).

La charge CT est bien représentative de la charge future de S3.

Chaque courant se compose d'une succession de petits travaux (séries) dont le temps maximum d'exécution peut aller jusqu'à quinze minutes dans des conditions très défavorables.

Les programmes à compiler et à exécuter sont en général assez courts. Un seul comporte plusieurs routines. Leur ensemble se rapproche d'une utilisation par des étudiants avec quelques demandes de chercheurs. On y trouve des petits programmes pour de simples problèmes (calcul de moyenne), des programmes exécutant de nombreuses entrées/sorties, des programmes orientés calcul.

De temps à autre s'exécute un programme basic afin de tenir compte de demandes particulières éventuelles. Les autres programmes sont écrits soit en fortran (travaux d'étudiants), soit en C (caractéristique d'UNIX).

Chaque série comporte une partie éditeur, ce qui correspond bien à l'emploi de l'interactif.

Presque toutes les séries demandent une sortie sur imprimante comme c'est souvent le cas pour garder une trace matérielle des fichiers.

La charge se base sur la "réalité" externe du système, en exécutant en parallèle des scenarii plausibles d'utilisation des terminaux.

UNIX possède tout une série de commandes qui ne sont pas reprises dans CT. Mais leur emploi et leur fréquence ne peuvent être connus étant donné les doutes sur l'utilisation de S3.

Cependant quelques caractéristiques d'UNIX se retrouvent dans le modèle complet de la charge, et ce, parce qu'elles jouent un double rôle. Elles font partie aussi bien du moniteur de test que de la charge elle-même (cfr V.2.4 "overhead").

En conclusion, c'est une charge qui respecte les conditions initiales (cfr la charge de S3), qui représente une "réalité" externe plausible, et dont les éléments principaux sont issus d'une étude de charge universitaire.

V.1.7 Rédaction de la charge CT.

Par commodités pour la suite, la masse de travail d'un terminal est appelée courant. Chaque courant comprend l'exécution de vingt séries (cfr V.1.6 Transformation de la charge).

Chaque courant se trouve dans un directoire particulier avec ses différents fichiers de travail. La description finale et détaillée d'un courant se trouve à l'annexe A.5. Les autres sont simplement présentés car ils sont structurés de la même façon.

En général, ils sont la simple traduction des séries interactives de B10 dans le langage de commande . Cependant, certains petits détails sont l'objet de modifications. L'éditeur de UNIX ne travaille pas par colonne. L'emploi d'une méthode qui simule l'interactif, pour le déroulement de la charge (cfr V.2.1 Lancement de la charge), oblige à tenir compte de certaines particularités du langage de commande, notamment pour la commande "sh" du "SHELL" (cfr III.2.2 Quelques caractéristiques).

V.2 Le moniteur de test.

Le mot "moniteur" est employé ici dans le sens le plus large. Il ne s'agit pas ici d'une construction interne au système dont le but est de collecter une trace d'un événement particulier mais bien d'un outil de mesure qui utilise uniquement les propriétés du "SHELL" (cfr III.2 Le SHELL) pour fournir une trace des résultats des mesures.

Une session est le temps mis par le système pour l'ensemble de la charge.

V.2.1 Lancement de la charge.

Le chapitre IV, Théorie de la charge, expose différentes méthodes employées pour simuler l'activité des terminaux en interactif.

L'emploi de personnel est impossible dans notre cas, d'abord parce que nous ne sommes que deux, ensuite parce que tout doit se faire automatiquement.

Il faut aussi rejeter l'emploi de terminaux à cassette car il n'y en a pas de disponibles.

Cependant, il est possible de travailler sur un PDP 11/45 et un PDP 11/70. Eventuellement la 11/45 peut être testée en lançant tout de la 11/70 qui, ensuite, collecte et traite les temps de réponse. Mais une tentative échoue. De plus, par la suite, nous remarquons que l'accroissement de la température dans la salle des machines, lorsque toutes deux travaillent, provoquent des pannes systèmes.

La seule méthode qu'il nous reste est le "BIBO" - - Batch in Batch out - Son utilisation dans notre cas, revient à simuler par un moniteur logiciel l'envoi et le retour de messages simultanément sur six vidéos. Le moniteur doit pouvoir s'adapter à la charge, collecter et stocker les mesures.

Envoi des messages.

Il est décidé que toute session se fait à partir de la console pour en conserver une trace écrite. C'est donc la console qui lance le début de la mise en marche de la charge. Par une propriété d'UNIX, nous lançons en parallèle plusieurs commandes et par extension plusieurs ensembles de commandes (cfr II.2.1 Quelques particularités).

Dans six fichiers se trouvent les différents courants et la console demande l'exécution simultanée de ces six fichiers par la"commande" :

sh flow 1 & sh flow 2 & sh flow 3 & sh flow 4 & sh flow 5 & sh flow 6,où flow j représente le fichier contenant le j^{me} courant.

Mais cette facilité cause une perte de travail au niveau du gérant des periphériques. La gestion de réception des messages envoyés des vidéos est réduite à sa plus simple expression. (La possibilité d'éviter cette perte en partie existe mais provoque un accroissement trop considérable de l' "overhead" pour être mise en pratique).

Il est aussi évident que la commande de lancement n'implique pas la même gestion des différents courants que s'ils parviennent réellement des terminaux. Dans la réalité les commandes sont prises en main dans l'ordre dans lequel elles arrivent dans le système. Dans la simulation cet ordre dépend essentiellement de la manière dont sont gérés les travaux par le système. Mais plus loin il est montré que cet inconvénient est finalement négligeable (cfr V .2.3 Les mesures).

Retour des messages.

La plupart des commandes de la charge demandent une réponse. Stocker ces réponses dans un fichier "poubelle" force à réduire le rôle du gérant des périphériques à néant. Pour contrer cet inconvénient nous dirigeons les réponses vers les vidéos concernés.

Grâce au signe d'indirection un message se déroute de sa sortie standard vers un fichier quelconque à déterminer. Or tout périphérique est considéré dans UNIX comme un fichier.

La simulation ne provoque aucune modification du traitement des commandes. Elle se rapproche d'assez près de la réalité pour les retours de message mais elle ignore le rôle du gérant des périphériques dans la gestion de réception des messages.

Cependant, contraitement au "BIBO" qui simule totalement la présence de terminaux, la méthode décrite emploie des terminaux concrètement.

Cette présence implique une initialisation des vidéos. Il faut les connecter au système.

Login : guest.

Afin que les réponses puissent apparaître convenablement sur les vidéos, il faut qu'ils soient connectés. Le fait de les connecter en "guest" est dû au souci de les placer exactement dans les conditions d'emploi par un utilisateur normal (cfr VI.3.3 Procédure Automatique).

V.2.2 Déroulement de la charge.

. Initialisation (cfr annexe A5).

La commande de lancement des courants s'exécute dans le directoire "/apm". Or le contenu des courants et de ses fichiers de travail se trouve dans d'autres directoires :

/apm/bench/tj : directoire pour le courant j.

Aussi l'ordre de lancement est en fait, dans "/apm" :

```
sh /apm/bench/t1/flow 1 &
sh /apm/bench/t2/flow 2 &
sh /apm/bench/t3/flow 3 &
sh /apm/bench/t4/flow 4 &
sh /apm/bench/t5/flow 5 &
sh /apm/bench/t6/flow 6 &
```

Et pour que les courants puissent se dérouler sans problème, ils doivent commencer par demander un changement de directoire. Ainsi a-t-on dans flow j ($1 \leq j \leq 6$), la commande de début :

chdir /apm/bench/tf (cfr III.2.2 Quelques particularités).

. Simulation des temps de réflexion.

Dans un scénario interactif, il est nécessaire de contrefaire les temps de réflexion que marquent un utilisateur. La seule possibilité offerte est l'emploi de la

commande : sleep <N secondes>
Elle provoque une attente de N secondes avant de passer à la commande suivante. Cependant elle crée un processus qui reste actif pendant N secondes. Elle occupe donc certaines ressources alors qu'en réalité aucune ressource n'est occupée pendant un temps de réflexion.

. Correction de la charge pendant l'exécution.

La répétition d'une même série dans un courant, implique qu'à chacune de ses exécutions, elle soit dans les conditions initiales adéquates. Si durant son développement elle modifie un fichier, il faut, lorsqu'elle se termine, remettre ce fichier en état pour la prochaine modification de ce fichier.

Un exemple est la série afb2 qui modifie un fichier (afpi2.f), le compile puis l'exécute. Cette série est appelée trois fois dans chaque courant. Si à la fin de la première exécution de afb2, afpi2.f n'est pas remis à son état initial, à la deuxième, les modifications ne peuvent être faites et on sort de afb2 (cfr II.2.2 Quelques caractéristiques). C'est pourquoi à la fin de la série afb2 se trouve une commande copiant dans afpi2.f le fichier initial dont une image existe sous un autre nom.

N.B. Cette copie peut être évitée en dupliciant deux fois l'image de la série et de ses fichiers de travail mais c'est perdre un point de contrôle pendant l'exécution du test (cfr V.2.6 Les tests).

De plus lorsque la session se termine, il se peut que l'arrêt se produise au milieu d'une (ou plusieurs) série(s), empêchant ainsi une mise à jour éventuelle de fichier. C'est pourquoi juste avant le lancement d'une autre session, une mise à jour de tous les fichiers se déroule par la commande :

sh majfb14 (cfr annexe A5).

. Particularité de la commande "sh".

Cette commande permet d'exécuter toute une suite d'autres qui se trouvent dans un fichier. Mais dès qu'une de celles-ci échoue, elle ne poursuit pas l'exécution des suivantes. Or certaines compilations doivent fournir un diagnostic d'erreurs pour respecter les scenarii de la charge. C'est pourquoi nous plaçons ces compilations dans des fichiers isolés que nous faisons exécuter dans les séries par l'emploi d'un nouveau "sh" (cfr III.2.2 Quelques particularités).

Si au niveau de la charge, la logique est respectée, il y a un apport supplémentaire de travail au niveau système. En bref, à la place d'une commande (compiler un programme), il y en a deux (exécuter les commandes d'un fichier et compiler un programme).

V.2.3. Les mesures.

Il est décidé que les mesures à entreprendre sont celles de temps de réponse (cfr I.2.1 Choix de l'objet des mesures).

UNIX possède une commande qui permet d'obtenir trois temps différents pour une autre commande qui s'exécute. Il s'agit de la commande "Time" (cfr III.2.2 Quelques caractéristiques).

Parmi les trois durées fournies seule la nommée "real" se rapproche d'une des définitions exposées dans I.5 Définitions des Performances de S3 . En effet, c'est le temps de cycle moins les temps de réflexion et de transmission utilisateur. Cependant, on peut tenir compte du temps de réflexion en comptant avec la commande "sleep".

Soit la séquence :

```
sleep 3
time <commande>
```

Le temps "real" de "time" ajouté à 3 donne ce que nous appelons le temps de cycle restreint. C'est ce temps que l'on mesure.

Remarquons qu'une propriété d'UNIX permet de se passer d'un gonflage du moniteur de test et d'une forte augmentation de l' "overhead" car son existence implique un emploi dans l'utilisation normale d'UNIX. Bien sûr, par leur fréquence élevée, les "times" provoque un certain "overhead" malgré tout.

Une fois les temps de réponse collectés, il faut les stocker dans un fichier particulier. Cette opération se déroule sans problème en employant l'indirection ">>" (cfr III.2.2 Quelques caractéristiques).

Mais faut-il pratiquer les mesures sur tous les courants ou sur un seul ?

. Tous les courants.

On a l'avantage de disposer d'autant de mêmes mesures pour une session qu'il y a de courants mais on a aussi un moniteur de test donc un "overhead" plus important. De plus un problème technique apparaît.

Supposons quatre courants - soit quatre vidéos ! Il faut que chacun se termine entièrement au moins une fois. La probabilité qu'ils se terminent tous au même moment est proche de zéro. Si lorsqu'un premier courant se finit, on ne le relance pas, la concurrence au niveau système se réduit d'une charge-vidéo et les temps restant à collecter pour les trois autres courants sont biaisés par rapport aux mêmes temps du premier. Il faut donc prévoir une boucle pour chaque courant. Et tous les courants s'arrêtent alors lorsque le dernier termine. Schématiquement :

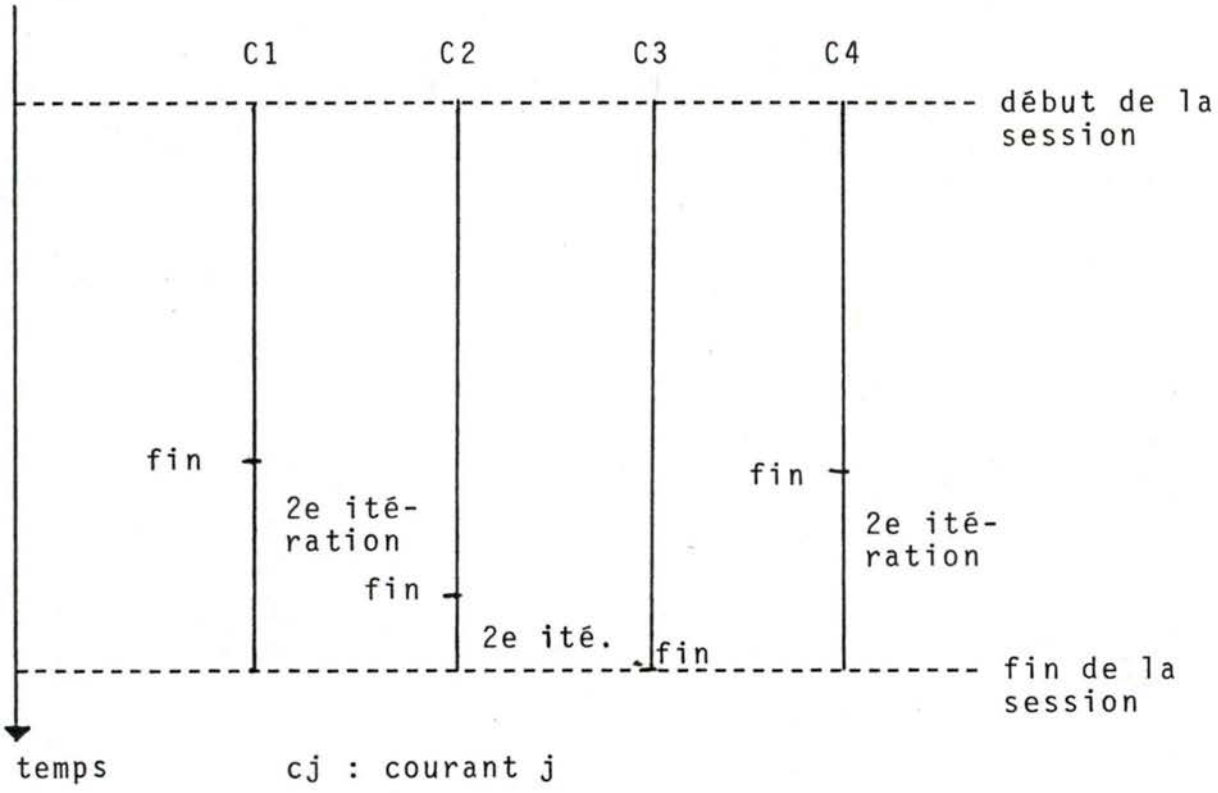


Fig. II.1 : Evolution temporelle des courants

Mais on ignore à priori quel courant finit le dernier si bien que le moniteur de test doit vérifier à chaque fin de courant si les trois autres ont fini et dans ce cas, arrêter la session sinon la laisser continuer.

Un problème de synchronisation - les deux derniers courants finissent presque ensemble -, une modification de la concurrence pendant la session - à chaque fin de courant le moniteur intervient -, un accroissement inévitable de l' "overhead" conduisent à un prix fort élevé pour plusieurs séries de temps sur une même session.

. Un seul courant.

Tous les inconvénients du cas précédent disparaissent. En effet, il suffit de mesurer sur un seul courant et quand il finit, on arrêtela session. Les autres courants bouclent sans cesse sur eux-mêmes jusqu'à ce que le courant testé les arrête. Bien sûr on ne récolte qu'une série de

mesures, mais si l'expérimentateur en désire plus, il peut relancer ce même test autant de fois qu'il le souhaite.

La façon dont le courant de test stoppe les autres est expliquée dans VI.3.3 Procédure Automatique .

L'ordre des commandes lancées par les différents courants importe moins. En effet, en ne testant qu'un seul courant, on peut ne considérer que celui-là; les autres forment une charge quelconque. C'est comme si seule l'utilisation d'un vidéo est prise en compte alors que ce qui se passe sur les autres n'intéresse pas. Dès lors, on ne peut prévoir l'ordre de succession des commandes au niveau système. Il peut être quelconque. Cependant on sait que la concurrence des autres courants suit les principes de la charge de S3.

Il faut être sûr que le courant de test commence lorsque les autres sont déjà en cours afin que les premières mesures soient faites avec la concurrence des autres courants. On retarde un peu, dans ce but, l'envoi du courant de test par l'emploi de la commande "sleep".

V.2.4 L' "overhead".

L' "overhead" est le travail qu'impose le moniteur de test. C.A. ROSE écrit qu'une sélection raisonnable d'événements et la quantité de données à enregistrer pour des applications dans un modèle de réseau à files d'attente, devrait être obtenue par un moniteur"à événements" qui provoquerait un "overhead" compris approximativement entre 5 et 10% (cfr /24/ C.A. ROSE).
Un moniteur "à événements" est, en gros, un moniteur software qui, dès qu'un type d'événements se produit, en enregistre sa trace.

Bien sûr, le but poursuivi ici n'est pas l'étude des files d'attentes et il ne s'agit pas d'un moniteur software. Cependant, cette remarque rentre bien dans le cadre

des mesures sur S3.

De plus, P. De RIVET estime qu'un "overhead" proche de 10% donne de bonnes mesures (cfr /8/ P. De RIVET).

Le gros du moniteur de test employé sur S3 comporte la collecte des mesures (commande "time") et stockage (signe ">>").

Le reste est réservé à assurer un bon déroulement de la charge - commandes "chdir" et "cp" - (cfr V.2.1. Déroulement de la charge) et à simuler au mieux l'emploi des vidéos - commandes "sh" et "sleep" - (cfr V.2.1 Déroulement de la charge et III.2.2 Quelques caractéristiques).

Finalement on s'aperçoit que le moniteur de test n'est constitué que de commandes UNIX. Le fait qu'elles existent implique un emploi par les utilisateurs. On peut donc dire qu'elles rentrent dans la charge CT en tant qu'éléments singuliers, sans constituer un véritable "overhead". Mais il ne faut pas oublier que leur fréquence est peut-être trop forte par rapport à la charge future surtout pour les commandes "time", ">>", "cp" et "sleep".

Dans le calcul de l' "overhead", on peut éliminer les "sleep" (cfr V.2.5 Révision de la charge).

En considérant tous les "time", ">>", "cp" comme l'overhead, les autres caractéristiques faisant partie de la charge singulière, on arrive au résultat de $\pm$ 10% d' "overhead" au point de vue temps d'exécution.

Exemple.

Si une session est le temps mis par le courant de tests pour se dérouler, alors une session comprenant quatre courants dont un seul sur lequel on mesure, prend 50'04" avec NBUF, NEXEC et SSIZE=SINCR ayant respectivement les valeurs 30, 2, 10.

Si une session est le temps que met le courant le plus lent pour se dérouler, une session sans mesure met

46'10", toujours avec les mêmes valeurs. Mais dans ce temps se trouve toujours les durées d'exécution des "cp". Dans la première expérience on a calculé qu'ils mettait en tout 1' pour s'exécuter. Donc une session sans "time", ">>", "cp" prend environ 45'10".

Il y a donc un écart de 4'54" qui correspond à 9,5% du temps mis pour une session avec mesure.

D'autres calculs sont présentés en annexe (Annexe A4) et permettent de constater que l' "overhead" en temps ne dépasse jamais 11%.

Evidemment ces chiffres ne sont qu'indicatifs de l'importance de l' "overhead" mais inclinent à penser que le moniteur de test est valable.

Il ne faut pas oublier que l'on peut mettre une partie des "time", et des ">>" dans la charge singulière, ce qui entraîne une diminution de l' "overhead".

V.2.5 Révision de la charge.

Certaines conditions du déroulement des tests ne sont guère favorables. Il faut rappeler que beaucoup de temps se passe à maîtriser toute une série de pannes principalement software - blocage du système - mais aussi hardware - Floating Point Processor -, si bien que, lorsque tout est prêt pour les tests, il ne nous reste plus beaucoup de jours.

Les premiers essais avec la charge construite sur six courants, montrent qu'un test met en moyenne deux heures pour s'exécuter entièrement. Une nouvelle série de pannes nous oblige à revoir la charge pour terminer dans les délais.

Diminuer la charge d'un courant revient à revoir toute la construction de celle-ci.

Mais on peut restreindre le nombre de courants à quatre par exemple. Cependant, une perte de concurrence en résulte.

Dans les courants restants se trouvent une longue succession de "sleep" qui simulent les temps de réflexion. Bien qu'ils entraînent la création de processus actifs, ils n'exigent pas énormément de travail de la part du système. En fait la concurrence qu'ils offrent est plus faible que celle d'autres commandes. En supprimant tous les "sleep" nous compensons une certaine partie du travail perdu en d'assez nombreux instants de la session.

Ainsi la réalité décrite par la charge devient moins proche d'un scénario interactif - il n'y a plus de temps de réflexion - mais maintient un certain volume de travail en concurrence au sein du système.

La pensée - toujours dans le but de gagner du temps - de tester la 11/70 à la place de la 11/45 est rejetée car l'expérience montre que le gain de temps pour un même test est de 4%.

La suite des événements lors des tests n'a fait que renforcer notre conviction de recourir à un tel raccourci pour respecter les délais.

Il reste que le temps de cycle restreint s'amoindrit du temps de réflexion.

V.2.6 Les tests.

Au cours des expériences, des opérations non prévues dans la charge se produisent - modification du mode d'un fichier -, que d'autres, prévues, ne s'effectuent pas - diagnostic : Command not found -.

De tels événements rendent le déroulement du test biaisé par rapport à ceux qui se terminent sans problèmes; ils obligent à recommencer le test.

La conséquence est que nous devons surveiller chaque session et dès qu'une anomalie apparaît, l'arrêter, recontrôler les différents travaux de la charge, corriger éventuellement puis relancer.

Après des essais préalables et réussis avec l'outil automatique, il apparaît que l'utiliser est s'exposer à le faire travailler inutilement pour toute une suite de tests biaisés et à tout recommencer. Le temps limité, interdit un tel risque. Chaque test s'opère donc semi-automatiquement et pendant leur déroulement des contrôles réguliers sont pratiqués : vérifier si tous les temps collectés sont bien stockés, regarder si aucun fichier de travail n'est altéré, etc...

Pendant une partie des tests, toute la boucle d'un test se fait sur la 11/45. Elle se compose des modules suivants (cfr VI.3.4 La boucle de contrôle) :

A - Modifier les paramètres
  - Compiler le nouveau système
B - Charger le nouveau système
C - Lancer la session de test

Le semi-automatisme se décrit comme suit :

- lancer manuellement l'exécution automatique de A et B
- Quand A et B sont finis, opérer certains contrôles
- Lancer manuellement la session de C

Par la suite, la disponibilité de deux disques et de deux PDP a conduit à raccourcir la boucle du temps. En effet, l'exécution du module A peut se faire indépendamment des modules B et C.

Finalement la boucle se déroule de cette facon :

initialisation charger un nouveau système sur le disque i sur la 11/45

corps: exécuter en parallèle

sur la 11/70	sur la 11/45
le module A sur le disque j	les modules BetC sur le disque i

intervertir les disques : i → j
j → i

si c'est le dernier test, aller en fin
sinon aller en corps

fin : exécuter le dernier test sur le 11/45

V.3 Conclusion.

La charge construite, bien que modifiée, reste une charge représentative et reproductible. Toutes les modifications sont considérées théoriquement.

L' "overhead" mesuré sur les temps de session a une importance d'environ 10%.

Le fait de ne pas employer l'outil automatique pour l'ensemble des tests n'est pas dû à son mauvais fonctionnement - il est au point ! - mais à la nécessité de surveiller de près les sessions de test afin d'éviter des erreurs répétitives inopinées et de gagner du temps pour respecter les délais déjà allongés à leur extrême limite.

Chapitre VI

MISE EN OEUVRE DE MESURES DE PERFORMANCES DE UNIX

Jean-Paul ADANS

Nous avons défini au chapitre I les performances que nous voulons mesurer. La charge est établie et avec elle, la technique de mesure proprement dite (commande "time"). Nous allons maintenant décrire le schéma global dans lequel s'insèrent ces mesures, ainsi que la méthode utilisée.

Nous parlerons brièvement des programmes que nous avons réalisés, et qui assurent le déroulement complet des mesures de manière automatique.

En premier lieu, rappelons la modélisation.

VI.1 Modèle métrologique.

VI.1.1 Les performances.

Nous avons dit que ce qui nous intéresse, c'est le temps de cycle restreint ou temps d'exécution. Comme une session de mesures fournit 118 données, il faut réduire ces grandeurs à une seule, de manière à obtenir une performance unique pour l'ensemble de la session.

Cette réduction est effectuée en calculant la moyenne arithmétique des 118 temps. Nous appellerons dès lors la performance considérée "temps d'exécution moyen".

On aurait pu considérer uniquement la somme des temps de réponse, mais la moyenne est tout aussi représentative et procure l'avantage de manipuler des grandeurs numériquement plus petites.

Le calcul de temps moyen d'exécution par classes de travaux ne se justifie que si notre but avait été la mesure des performances de certaines facilités de UNIX (compilateur, assembleur, éditeur, ...).

Ce n'est pas le cas et dès lors nous nous contenterons de l'aperçu global fourni par le temps d'exécution moyen.

VI.1.2 Choix des paramètres.

La technique que nous avons mise au point et les programmes de contrôle qui la réalisent permettent l'analyse en fonction de paramètres définis par remplacement littéral dans le code source de UNIX. Cette opération se fait au moyen de la pseudo instruction "#define" du langage de programmation C.

Grâce à cette instruction, toute occurrence d'une certaine chaîne de caractère est remplacée par le second argument, également considéré comme chaîne de caractères.

Par exemple, grâce à l'instruction

```
#define    NBUF    15
```

toute occurrence de la chaîne "NBUF" sera remplacée dans le texte source par la chaîne "15".

Les paramètres qui seront à analyser doivent être définis de cette manière, c'est-à-dire qu'ils seront constants pour la durée de vie du système. De plus, ils doivent avoir une valeur numérique entière pour être acceptés par les programmes de contrôle.

Ceci n'empêche cependant pas l'utilisation de paramètres qualitatifs (choix d'algorithme par exemple). Ceux-ci devront être combinés avec des instructions conditionnelles dans le code du système d'exploitation.

Le texte source de UNIX est réparti sur plusieurs fichiers et est écrit pour 95% dans le langage de programmation C. Certains fichiers ne contiennent que des déclarations de variables et de constantes et sont reconnus par un nom se terminant par ".h". Les fichiers dont le nom se termine par ".c" contiennent le texte des procédures formant le système d'exploitation. Lors de la compilation des fichiers ".c", ceux-ci englobent les fichiers ".h" au moyen de la pseudoinstruction "#include" dont l'effet est l'insertion

VI.2 Définition du plan d'expérience.

VI.2.1 Paramètres.

Dans le paragraphe précédent, nous avons expliqué comment déterminer les paramètres.

Remarquons ici simplement que ces paramètres doivent avoir une valeur numérique.

VI.2.2 Valeurs.

Pour pouvoir décrire le plan d'expérience , il faut déterminer les valeurs des niveaux des paramètres ou facteurs.

Une bonne technique consiste à analyser les points où ces paramètres interviennent. Ceci peut conduire à déterminer la plage de variation de leur valeur.

Cela n'est pas toujours possible et il faut alors fixer arbitrairement les niveaux.

Nous avons adopté la détermination d'une variation unitaire. Celle-ci est fixée dans un rapport raisonnable avec l'ordre de grandeur du paramètre :

- 1 < paramètre < 10 : variation unitaire = 1
- 10 < paramètre < 50 : variation unitaire = 5
- 50 < paramètre <100 : variation unitaire = 10
- etc.

Rappelons que ce choix est arbitraire. Il peut être éventuellement affiné lorsque les résultats obtenus semblent en indiquer la nécessité.

VI.2.3 Plan d'expérience.

Le plan que nous avons choisi d'utiliser est un plan complet croisé. Cela signifie (Cf II.3 Plans d'expérience) que tous les niveaux de chaque facteur sont combinés avec tous les niveaux des autres facteurs.

La définition du plan d'expérience se fait à l'aide du programme "mkbench.c". Celui-ci questionne l'expérimentateur sur les paramètres et leurs valeurs :

- nombre de paramètres,
- noms des paramètres,
- nombre de niveaux pour chaque paramètre,
- valeur de chaque niveau.

Le programme réalise alors le croisement complet en produisant un fichier de valeurs des paramètres(fichier "values"). Ce fichier sera utilisé ultérieurement pour la modification automatique des valeurs.

Ce programme réalise également un fichier de spécifications nécessaire pour la suite ("params.c").

VI.3 Construction de l'outil de mesure.

VI.3.1 Description.

Nous avons dit comment les paramètres du système d'exploitation sont pris en compte.

Ceci implique qu'après chaque modification des valeurs, il faut régénérer le système en le compilant à nouveau, en arrêtant la machine et en rechargeant la version ainsi obtenue.

Etant donné d'une part, que la planification atteint rapidement une taille importante (3 facteurs à 3 niveaux chacun = 27 expériences multipliées par le nombre de

répétitions désirées; et, d'autre part, que le système UNIX est écrit en langage de haut niveau, il nous a semblé intéressant d'automatiser la procédure. De cette manière, son déroulement ne nécessite pas la présence de l'expérimentateur.

L'outil de mesure revêt donc la forme générale donnée par l'organigramme de la figure VI.1.

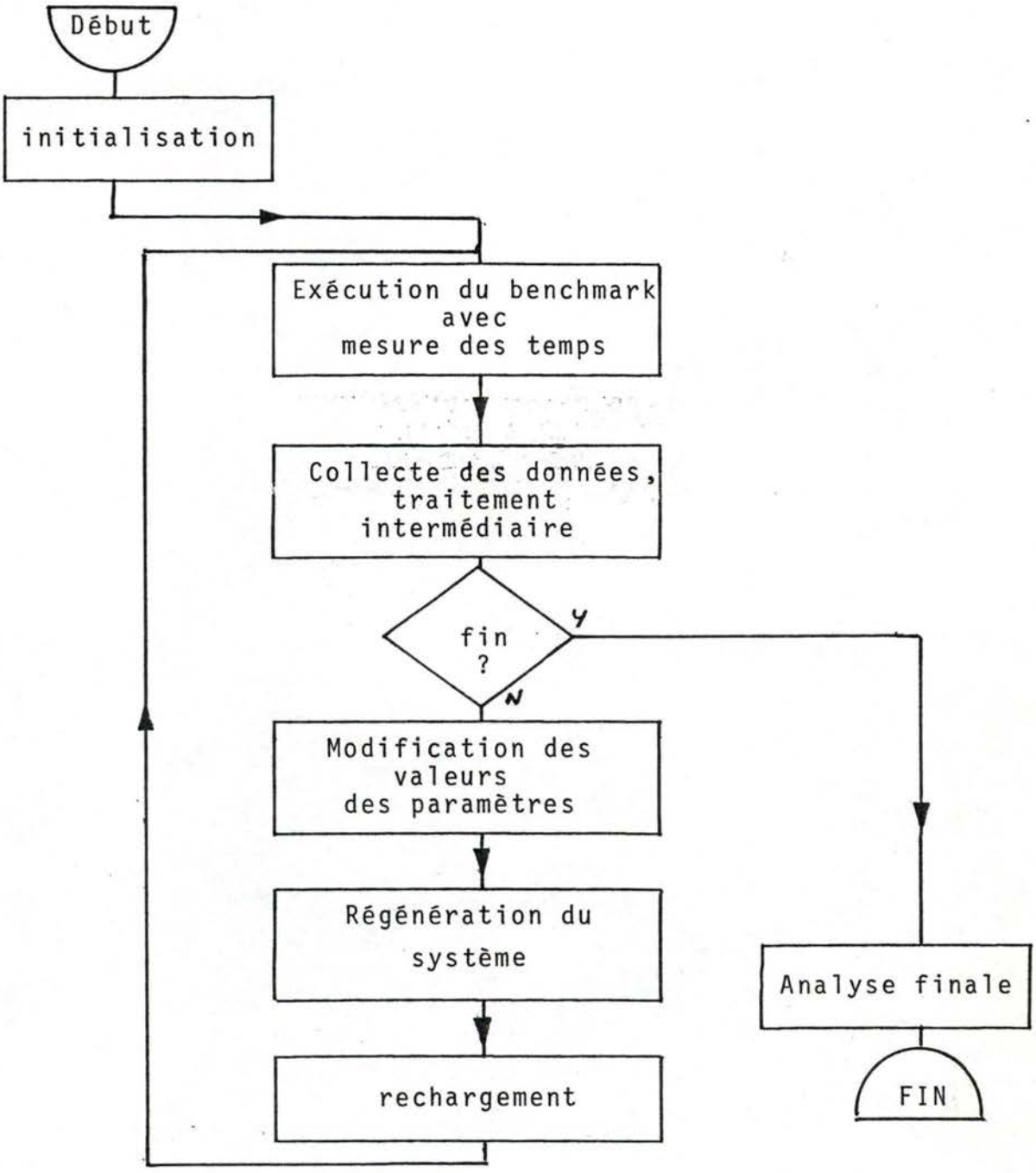


Fig. VI.1 Organigramme général de l'outil de mesure

Nous avons donc dû adapter le système d'exploitation afin de réaliser cet automatisme. Avant de décrire les modifications, il nous faut voir comment les actions s'enchaînent de manière manuelle.

VI.3.2 Procédure manuelle.

a. Chargement.

La première phase consiste à charger le système d'exploitation. Cela signifie transférer la version de UNIX adéquate, du disque (où elle constitue un fichier comme les autres), en mémoire centrale, et d'initialiser son exécution.

Pour ce faire, on introduit, grâce aux clés du panneau frontal de l'ordinateur, une adresse dans le compteur ordinal. Cette adresse est celle d'un petit programme de chargement (20 instructions), résidant en mémoire morte de la machine, capable de lire sur la périphérique. Il va de soi qu'il existe différents programmes pour les divers types de disques.

Ensuite, une pression sur la touche START provoque :

1) la mise à zéro des registres de segmentation et des registres de contrôle des périphériques,
2) l'exécution de la séquence d'instructions débutant à l'adresse contenue dans le compteur ordinal.

Par exemple, pour la machine que nous avons utilisée, le chargement débute par

```
LOAD ADDRESS 01773320
START
```

Cette séquence dépend, nous l'avons dit, du type de périphérique mais aussi de la machine utilisée.

L'effet du programme ainsi exécuté est la lecture du premier bloc de 512 octets sur le disque, et la copie de celui-ci dans la mémoire centrale (à partir de l'adresse

zéro). Ce bloc est supposé contenir une séquence d'instructions exécutables et le contrôle est transféré à ce programme.

Celui-ci est ce que nous appellerons un chargeur programmé (en contradiction avec le chargeur fixe en mémoire morte), qui connaît la structure des fichiers résidant sur le disque.

Lors de l'exécution de ce programme, celui-ci imprime un caractère '@' à la console de l'opérateur et attend de celui-ci l'introduction du nom d'un fichier, censé contenir le code exécutable de UNIX.

Lorsque ce fichier est trouvé, il est transféré en mémoire centrale et exécuté.

b. login.

UNIX effectue une série d'initialisations puis se subdivise en deux processus : l'un qui réalisera la gestion des processus, l'autre chargé de la question des lignes de communication.

Ce dernier envoie à tous les terminaux le message

login :

et attend l'introduction d'une identification d'utilisateur. Lorsque ceci se produit et si cette identification est correcte, l'exécution d'une instance du "SHELL" est provoquée et l'utilisateur peut dès lors travailler. Le "SHELL" signale cette aptitude par le caractère "%" ("#" pour le super-utilisateur (1)).

---

(1) voir aussi /27/ THOMPSON & RITCHIE; sec. VIII, boot procedures; sec. VIII, getty; sec. I, Login .

VI.3.3 Procédure automatique.

On voit apparaître les points où une adaptation est nécessaire :
- suppression de la frappe d'un nom de fichier après "@";
- suppression de la frappe d'une identification après "Login:";
- suppression de la frappe d'une commande après "%".

a. Chargement.

Le premier point est escamoté par la modification du chargeur programmé. Celui-ci chargera toujours le fichier "tunix". Il faut dès lors prendre soin à avoir la bonne version dans ce fichier.

b. Login.

Ce point peut être contourné de deux manières. Il est possible de charger UNIX en mode "mono-user", ce qui ne permet qu'à la console opérateur de travailler; le fonctionnement des autres terminaux étant inhibé dans le sens terminal (clavier) - ordinateur.

Ceci ne nous a pas paru indiqué, puisqu'un terminal non initialisé au moment du "login" ne permet pas l'impression lisible de textes. (Nous évitons ici des explications longues et fastidieuses pour le lecteur non intéressé. Pour plus de détails, nous renvoyons à /27/, THOMPSON & RITCHIE; sec. VIII, getty; sec. II, stty; et sec. I, login).

Une autre possibilité a été de provoquer automatiquement une identification en court-circuitant le "login" (cf annexe, programme getty.c modifié).

c. Commande.

En ce qui concerne ce dernier point, nous avons pu mettre à profit une caractéristique de UNIX, qui permet au gérant de définir des utilisateurs n'ayant pas accès à toutes les commandes. Disons simplement qu'au lieu de l'exécution du "SHELL" après le "login", un autre programme est invoqué. Il suffit alors d'indiquer un programme de contrôle tel la procédure "loop" expliquée au paragraphe suivant (voir aussi /27/, THOMPSON & RITCHIE, sec. V, /etc/passwd).

VI.3.4 La boucle de contrôle.

Le système est ainsi initialisé. Un terminal (nous avons choisi la console principale) est identifié avec un nom d'utilisateur provoquant l'exécution de la procédure de mesure. Les autres terminaux (dans notre cas 4) sont identifiés de manière à pouvoir recevoir les sorties des programmes constituants la charge.

La procédure de mesure est décrite en annexe. Elle porte le nom "loop". Elle reflète la structure générale de notre outil, et nous pouvons maintenant en donner l'organigramme exact (fig. VI.2).

Deux points cependant sont à remarquer. Il s'agit des commandes "skill" et "reboot", que nous avons implimentées au moyen de deux nouveaux appels au système.

"skill" organise la synchronisation des programmes de la charge, en arrêtant les courants exécutés en parallèle, avec le courant 1, sur lequel portent les mesures, après terminaison de celui-ci.

"reboot" provoque le rechargement du système, en exécutant un "reset" (mise à zéro des registres de segmentation et de contrôle des périphériques) et un saut inconditionnel à l'adresse du chargeur fixe.

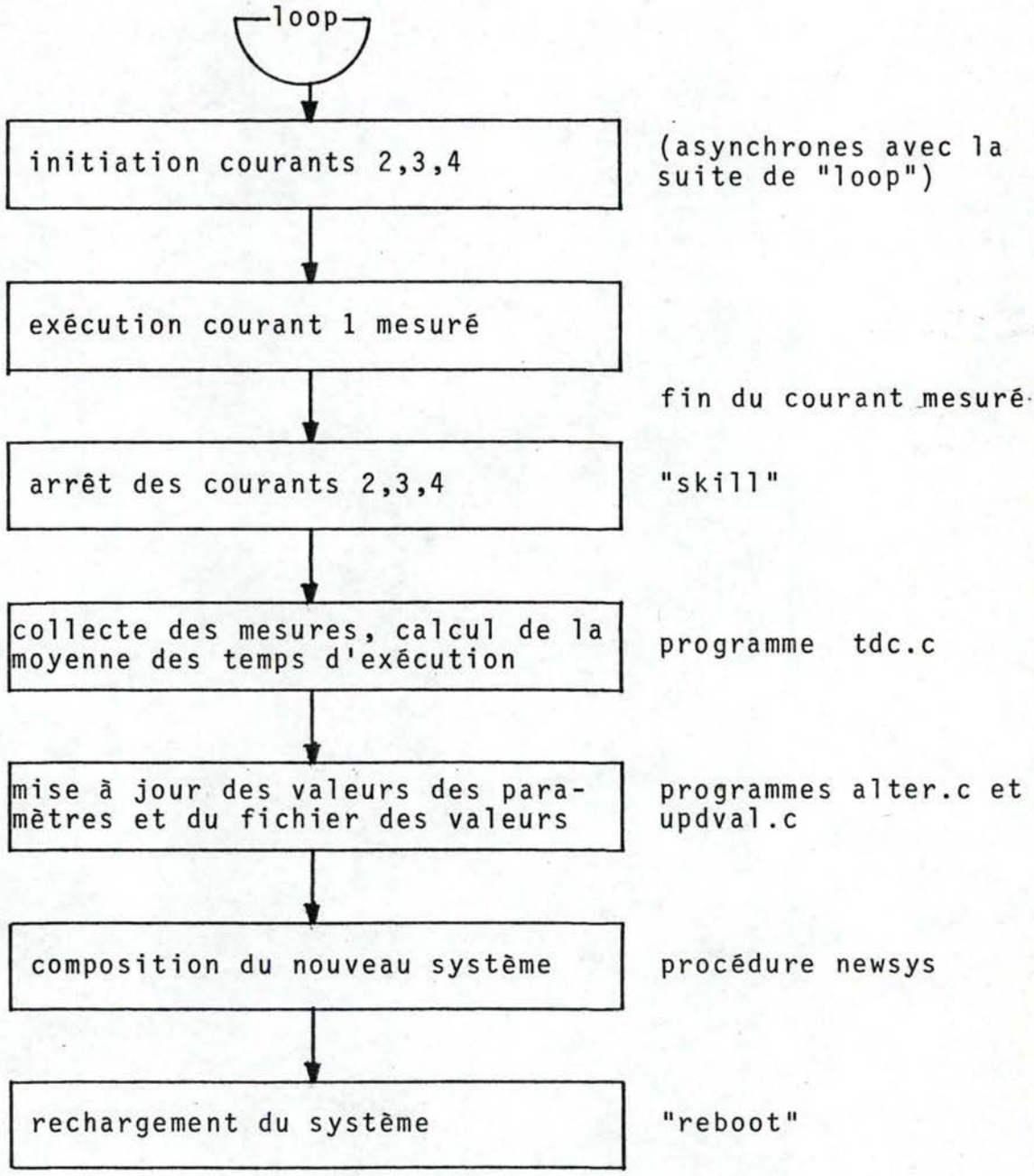


Fig. VI.2 Organigramme de la boucle de contrôle

VI.3.5 Collecte des données.

A la fin de l'exécution du courant 1, nous disposons d'un fichier contenant des messages de temps. Ce fichier contient également, inévitablement, quelques messages d'erreurs.

Le programme "tdc.c" réalise la lecture sélective de ce fichier et le calcul de la moyenne des temps d'exécution. Cette moyenne est ajoutée au fichier "means", qui sera lu par le programme d'analyse final.

VI.3.6 Mise à jour des valeurs des paramètres.

A ce moment, la première ligne du fichier "values" donne les valeurs des paramètres pour la génération courante du système. Le programme "updval.c" retire cette ligne du fichier. Si c'est la dernière, c'est-à-dire si le plan d'expériences est achevé, il provoque l'exécution d'une procédure d'arrêt de la boucle automatique, en réinstallant le système standard.

Si ce n'est pas la dernière ligne, le programme "alter.c" est exécuté. Celui-ci lit la première ligne du fichier restant en modifiant, en fonction des valeurs qu'il y a trouvées, le fichier "tune.h".

Le système peut alors être compilé et chargé à nouveau, ce qui est réalisé par la procédure "newsys".

Les textes de ces programmes et procédures se trouvent en annexe.

VI.3.7 Traitement des résultats.

Lorsque le processus de mesure est entièrement terminé, l'expérimentateur peut exécuter le programme d'analyse de variance "anova". Il s'agit en fait de plusieurs modules indépendants, enchaînés de manière à limiter au maximum la mémoire utilisée. Ceci est nécessaire car ces programmes mis ensemble risquent d'avoir une taille trop importante.

Du reste, ce découpage en phases permet aisément un accroissement des possibilités. En effet, la version actuelle permet l'analyse de plans complets de 1 à 5 facteurs, chacun à au plus 3 niveaux.

Pour augmenter le nombre de niveaux, il suffit de changer une valeur dans le fichier "spac.h".

Ce programme produit en sortie un tableau reprenant les observations classées par niveaux ainsi que la table d'analyse de variance.

VI.3.8 Sécurité de l'outil.

Il va de soi que cet outil de mesure est composé d'un grand nombre de fichiers et de programmes. Le nombre des entrées-sorties effectuées au cours d'une session de mesure est très important et il se peut que des erreurs se produisent.

Mises à part quelques sécurités assez classiques (copie des fichiers, protection d'accès), il semble que des programmes de vérifications devraient compléter l'outil si une exploitation régulière est envisagée. Dans notre cas, les copies et protections ont suffit à éviter des désastres (voir aussi V.2 Le montieur de test).

Il serait intéressant, dans la mesure des possibilités, de travailler avec un support secondaire, tel une

bande magnétique, pour enregistrer les précieux résultats des mesures.

Afin de nous garantir contre le maximum d'erreurs possible, nous avons incorporé des commandes de maintenance prévues par UNIX, tel la vérification des disques.

En effet, pour des raisons que nous n'avons pu éclaircir, des erreurs se produisaient à intervalles irréguliers sur les disques.

Enfin, n'oublions pas de mentionner une procédure de mise à jour des fichiers de la charge, ceux-ci étant modifiés au cours d'une session.

VI.4 Exemple complet.

Nous avons utilisé notre outil de mesure pour un plan complet de 3 facteurs à 3 niveaux.

VI.4.1 Choix des paramètres.

Il existe à profusion des paramètres sur lesquels on peut se baser pour des mesures de performance.

Ils se groupent en deux classes. Ceux dits quantitatifs sont des variables du système. Les qualitatifs représentent, concrètement, une partie du code.

Nous écartons ces derniers pour ne pas fausser les expériences par des considérations initialement erronés, car, lorsqu'on aborde un système, et même plus tard, il n'est pas facile de cerner l'importance relative des algorithmes et nous risquons de ne rien approter de neuf par l'étude de nos mesures.

Parmi les paramètres quantitatifs qu'offrent UNIX, certains jouent un rôle central. Ils interviennent en

général de près dans l'existence des processus.

Nous retenons essentiellement les paramètres suivants :

- SSIZE (cfr III.3.2 Description des composants d'une image d'un processus).

Quand un nouveau processus se crée, le système lui accorde en général une zone de pile par l'appel système "exec". D'office on lui attribue

SSIZE x 32 mots.

- SINCR (cfr III.3.2.1 Description des composants d'une image d'un processus).

C'est l'unité d'incrémentation de la pile. On l'augmente quant le pointeur de pile ne se trouve plus dans la zone de pile. D'office on ajoute

SINCR x 32 mots.

NPROC (cfr III.3.2.1 Description des composants d'une image d'un processus).

C'est le nombre maximum de processus qui peuvent tourner simultanément. Si un (NPROC + 1)^{me} processus arrive, il est rejeté.

NOFILE (Cfr III.3.4.2 Accès aux fichiers).

C'est le nombre maximum de fichiers qui peuvent être ouverts simultanément par un même processus. Un essai de dépassement de cette limite donne lieu à un rejet du processus.

NFILE (Cfr III.3.4.2 Accès aux fichiers).

C'est le nombre maximum de fichiers ouverts simultanément par tous les processus. Le processus qui tente de dépasser cette borne est rejeté.

Si on a

NFILE > NPROC x NOFILE

il n'y aura jamais dépassement.

NTEXT (Cfr III.3.2.1 Description des composants d'une image d'un processus).

C'est le nombre maximum de segments de texte. Tout essai de dépassement bloque le système, il faut le relancer depuis le début.

NEXEC (Cfr III.2.1 Les primitives).

C'est le nombre maximum d' "exec" simultanés. Une demande supplémentaire provoque une attente.

CMAPSIZ & SMAPSIZ (Cfr III.3.1.4 Allocation et Désallocation).

C'est le nombre maximum de trous mémoires (en mémoire centrale et sur disque) disponibles pour y placer un processus.

NINODE (Cfr III.3.4.3 Le tableau "Inode").

C'est le nombre maximum d' "Inode". Il en faut un par fichier actif, par fichier monté, par fichier de texte, par directoire courant, par "root". Un essai de débordement provoque le rejet du processus demandeur.

NMOUNT (Cfr III.3.4.1 Système de fichier).

C'est le nombre maximum de fichiers pouvant être montés. Une tentative de dépassement provoque le rejet du processus demandeur.

NBUF (Cfr III.3.3 Les tampons pour supports de type bloc).

C'est le nombre maximum de tampons disponibles pour des supports orientés bloc (ils peuvent servir à d'autres usages). Une demande supplémentaire provoque cette attente.

Si dans une session d'expérience, une commande du courant de test ne s'exécute pas parce qu'elle tente de forcer la valeur de certains paramètres (NPROC, NOFILE, ...) le temps de réponse collecté est nul, la série qui la contient risque de ne plus se poursuivre, les autres exécutions de cette série peuvent être faussées et finalement le temps mis par le courant est biaisé par rapport aux temps des expériences exécutées parfaitement.

Ces paramètres ne sont intéressants que pour connaître la limite inférieure au-delà de laquelle sont rejetés régulièrement des processus. Une telle recherche est secondaire dans nos mesures et nous n'avons pu la faire par manque de temps.

De plus, quand ces paramètres ont une valeur suffisante pour qu'il n'y ait pas de rejet, rien ne sert de les augmenter puisque la borne qu'il détermine n'est jamais franchie. Et ce n'est pas pour quelques unités au-dessus de leur limite inférieure, qui coûte en ressource au système. Elles font juste occuper quelques centaines de Bytes à des zones inutilisées de la mémoire principale.

C'est pourquoi sont éliminés les paramètres NPROC, NFILE, NOFILE, NTEXT, NINODE, NMOUNT.

Quant à faire varier CMAPSIZ ou SMAPSIZ, ce n'est guère intéressant car quel directeur de centre leur donnerait une valeur tel qu'ils ne pourraient représenter tous les trous mémoires ? Et pourquoi les augmenter si on est sûr que chaque trou sera pris en compte ? Ce qui est le cas quand SMAPSIZ, CMAPSIZ$\geq$2xNPROC (l'explication est fastidieuse elle ne peut être développée ici).

Il reste donc SSIZE, SINCR, NBUF et NEXEC. Nous avons décidé de fusionner les deux premiers en un seul (SSIZE=SINCR) appelé incrément de pile pour pouvoir tenir compte des quatre, car le temps nous a forcé à ne considérer que trois paramètres.

Les autres paramètres sont laissés aux valeurs de la version standard de UNIX. On peut ainsi supposer que la place inutile qu'ils font occuper est assez minime par rapport à l'ensemble de la mémoire physique.

II.4.2 Plan d'expériences.

Les valeurs standards de ces paramètres sont données au tableau VI.1

Paramètres	Valeur
NBUF	15
NEXEC	3
SSIZE/SINCR	20

Tableau VI.1 Valeurs standards des paramètres.

D'après le principe de la variation unitaire, nous avons déterminé pour chacun des paramètres trois niveaux.

Une série d'expériences préalables portant uniquement sur le paramètre NBUF, nous a conduits à ne retenir pour celui-ci que les valeurs supérieures à 15.

Le plan d'expériences se construit donc à l'aide du tableau VI.2

Facteur	Niveaux		
	0	1	2
NBUF	20	25	30
NEXEC	2	3	4
STACK	10	15	20

Tableau VI.2 Niveaux des paramètres.

La planification est donc constituée par un croisement de ces niveaux.

Il convient d'attacher une importance particulière au choix du nombre de répétitions. Dans notre cas, nous avons limité le plan à une seule répétition. Nous avons vu (II.4.3) que ceci revient à négliger d'office les interactions d'ordre le plus élevé (ici, les interactions d'ordre 3).

Avec la charge telle que nous l'avons construite, une session de mesure avec préparation du nouveau système, c'est-à-dire une boucle complète, s'exécute en 75 à 90 minutes. Etant donné une possibilité d'occupation de la machine limitée, nous avons été contraints de réduire ainsi à priori le nombre d'expériences.

L'extrait de protocole de la figure VI.3 montre l'exécution du programme "mkbench.c" déterminant ce plan d'expériences.

```
# mkbench
CROSSED DATA DESIGN:

How much factors ?                          3
Factor  B :     Name :                      NBUF
                How much levels ?           3
                        Level 0 :           Value ? 20
                        Level 1 :           Value ? 25
                        Level 2 :           Value ? 30
Factor  C :     Name :                      NEXEC
                How much levels ?           3
                        Level 0 :           Value ? 2
                        Level 1 :           Value ? 3
                        Level 2 :           Value ? 4
Factor  D :     Name :                      STACK
                How much levels ?           3
                        Level 0 :           Value ? 10
                        Level 1 :           Value ? 15
                        Level 2 :           Value ? 20

How much replications in this design ?  1


27 experiences are necessary for this design
mkbench done
#
```

Fig. VI.3 Exécution du programme "mkbench.c"

de définition du plan d'expériences

Le tableau VI.3 donne le contenu du fichier "values", reflétant le plan d'expériences.

NBUF	NEXEC	STACK
20	2	10
20	2	15
20	2	20
20	3	10
20	3	15
20	3	20
20	4	10
20	4	15
20	4	20
25	2	10
25	2	15
25	2	20
25	3	10
25	3	15
25	3	20
25	4	10
25	4	15
25	4	20
30	2	10
30	2	15
30	2	20
30	3	10
30	3	15
30	3	20
30	4	10
30	4	15
30	4	20

Tableau VI.3 Plan complet.

VI.4.3 Résultats des mesures.

Le tableau VI.4 donne une représentation des résultats des mesures.

numéro de l' expérience	valeur des paramètres NBUF	NEXEC	STACK	temps total (en sec.)	temps moyen (en sec.)
1	20	2	10	2925.00	24.788
2	20	2	15	2987.00	25.314
3	20	2	20	3086.00	36.153
4	20	3	10	2875.00	24.361
5	20	3	15	2960.00	25.085
6	20	3	20	2904.00	24.610
7	20	4	10	2969.00	25.161
8	20	4	15	3141.00	26.619
9	20	4	20	3132.00	26.542
10	25	2	10	2587.00	21.924
11	25	2	15	2729.00	23.127
12	25	2	20	2703.00	22.907
13	25	3	10	2578.00	21.847
14	25	3	15	2581.00	21.873
15	25	3	20	2655.00	22.500
16	25	4	10	2615.00	22.161
17	25	4	15	2661.00	22.551
18	25	4	20	2602.00	22.051
19	30	2	10	2414.00	20.458
20	30	2	15	2482.00	21.034
21	30	2	20	2438.00	20.661
22	30	3	10	2435.00	20.636
23	30	3	15	2433.00	20.619
24	30	3	20	2490.00	21.102
25	30	4	10	2428.00	20.576
26	30	4	15	2452.00	20.780
27	30	4	20	2470.00	20.932

Tableau VI.4 Résultats des mesures.

VI.4.4 Interprétation.

Une représentation graphique des résultats est donnée par les figures VI.4 à VI.9.

Légende des figures VI.4 à VI.9

- Le temps en ordonnée des figures VI.4, VI.6, VI.8 représente le temps global d'une session (en minutes)
- Le temps en ordonnée des figures VI.5, VI.7, VI.9 représente le temps moyen d'une commande (en secondes)
- Entre parenthèses sont indiqués les paramètres pris comme constante
- En abscisse sont représentées les valeurs du paramètre en fonction desquelles sont portés les temps sur le graphique

Fig. VI.4 Temps global en fonction de NEXEC

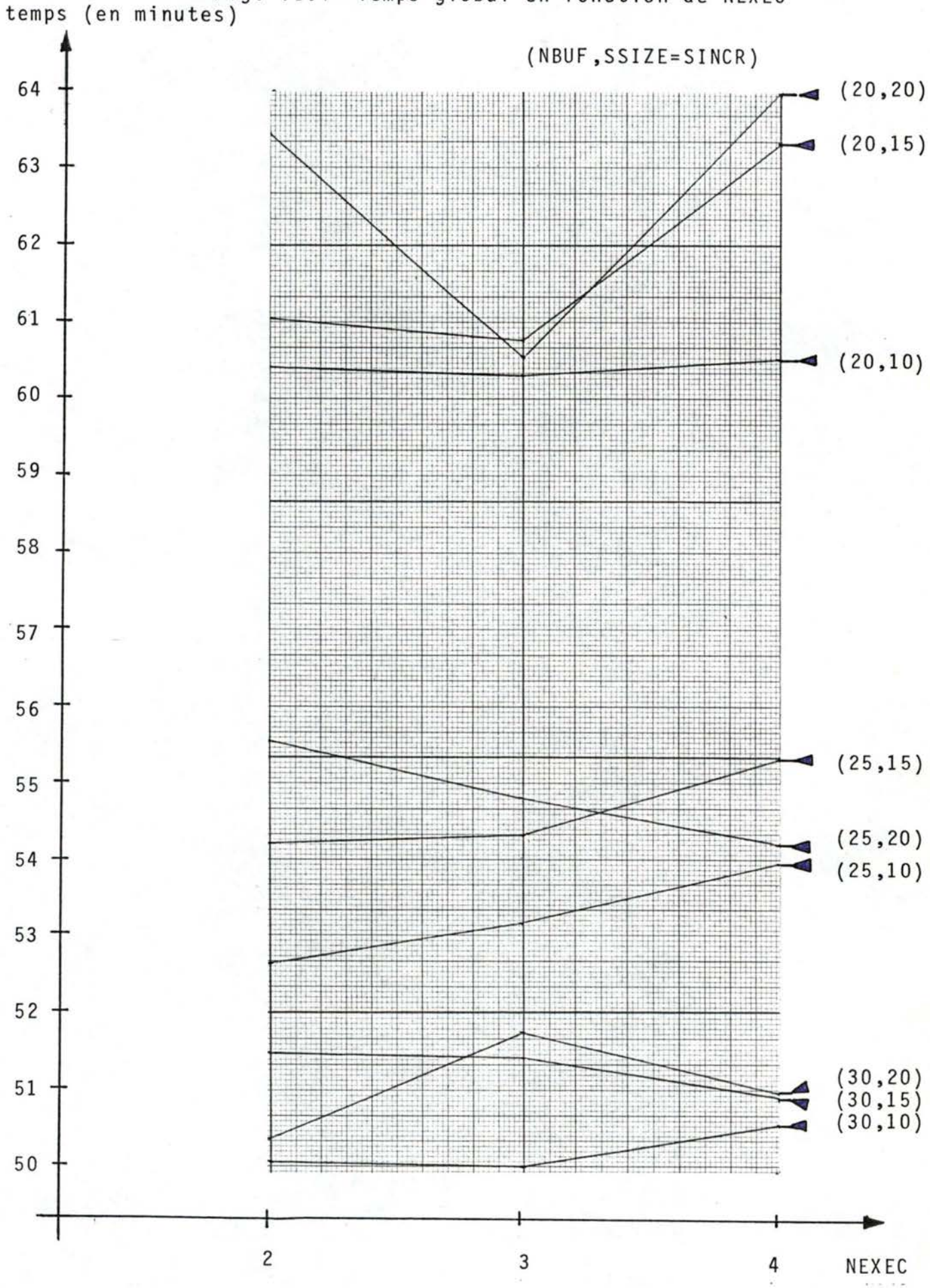


Fig. VI.5 Temps moyen en fonction de NEXEC

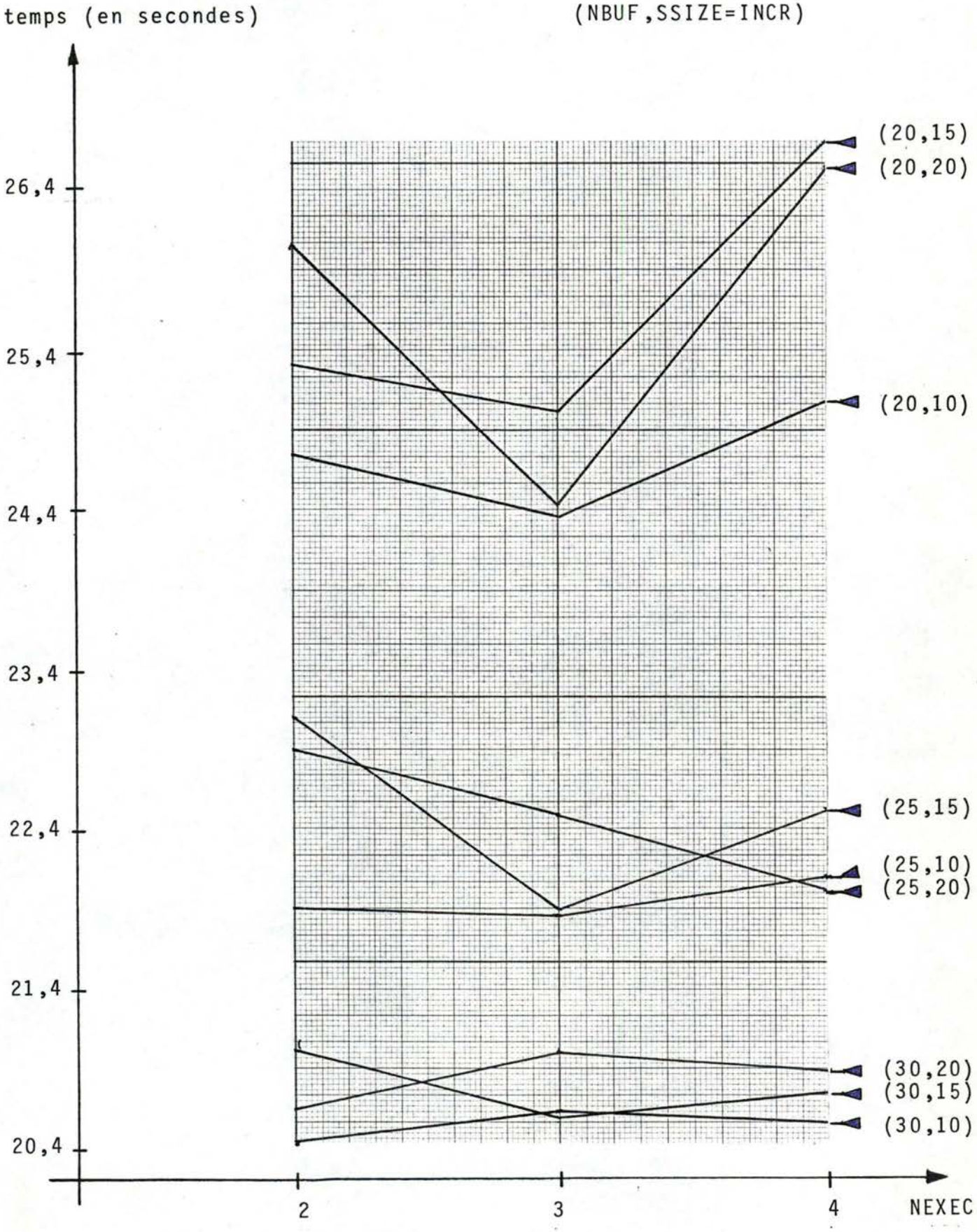


Fig. VI.6 Temps global en fonction de NBUF

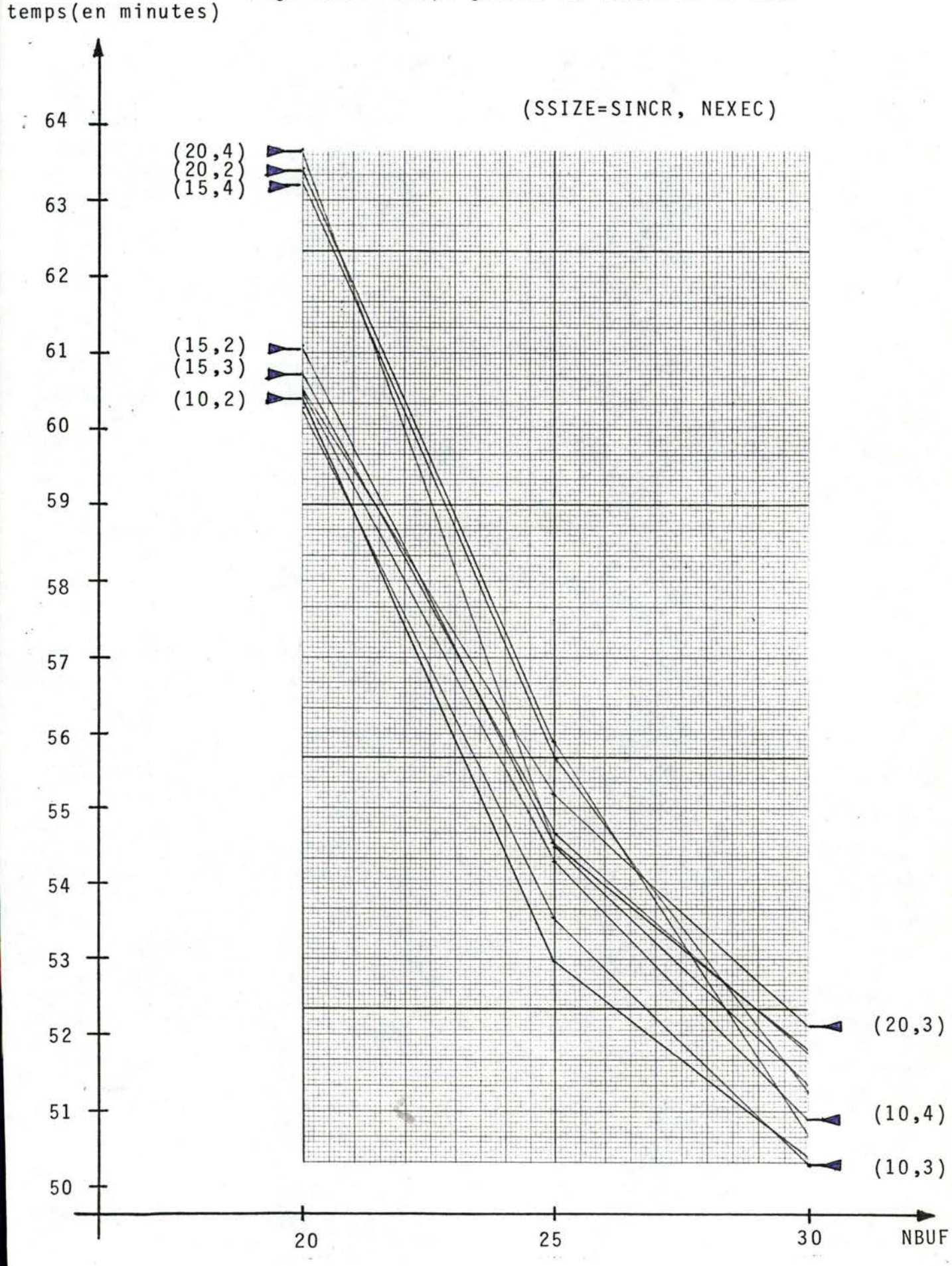


Fig. VI.7 Temps moyen en fonction de NBUF

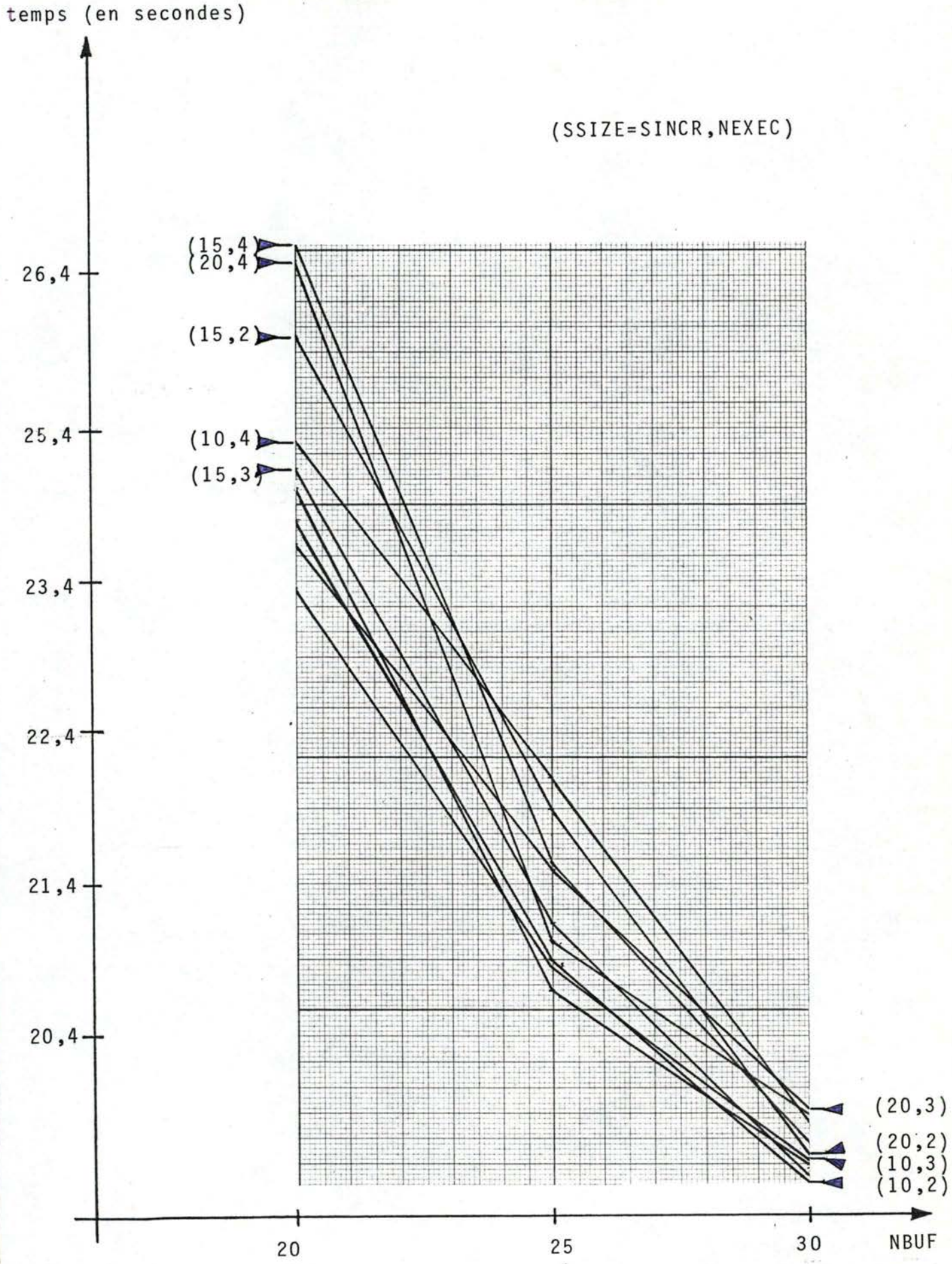


Fig. VI.8 Temps global en fonction de SSIZE=SINCR

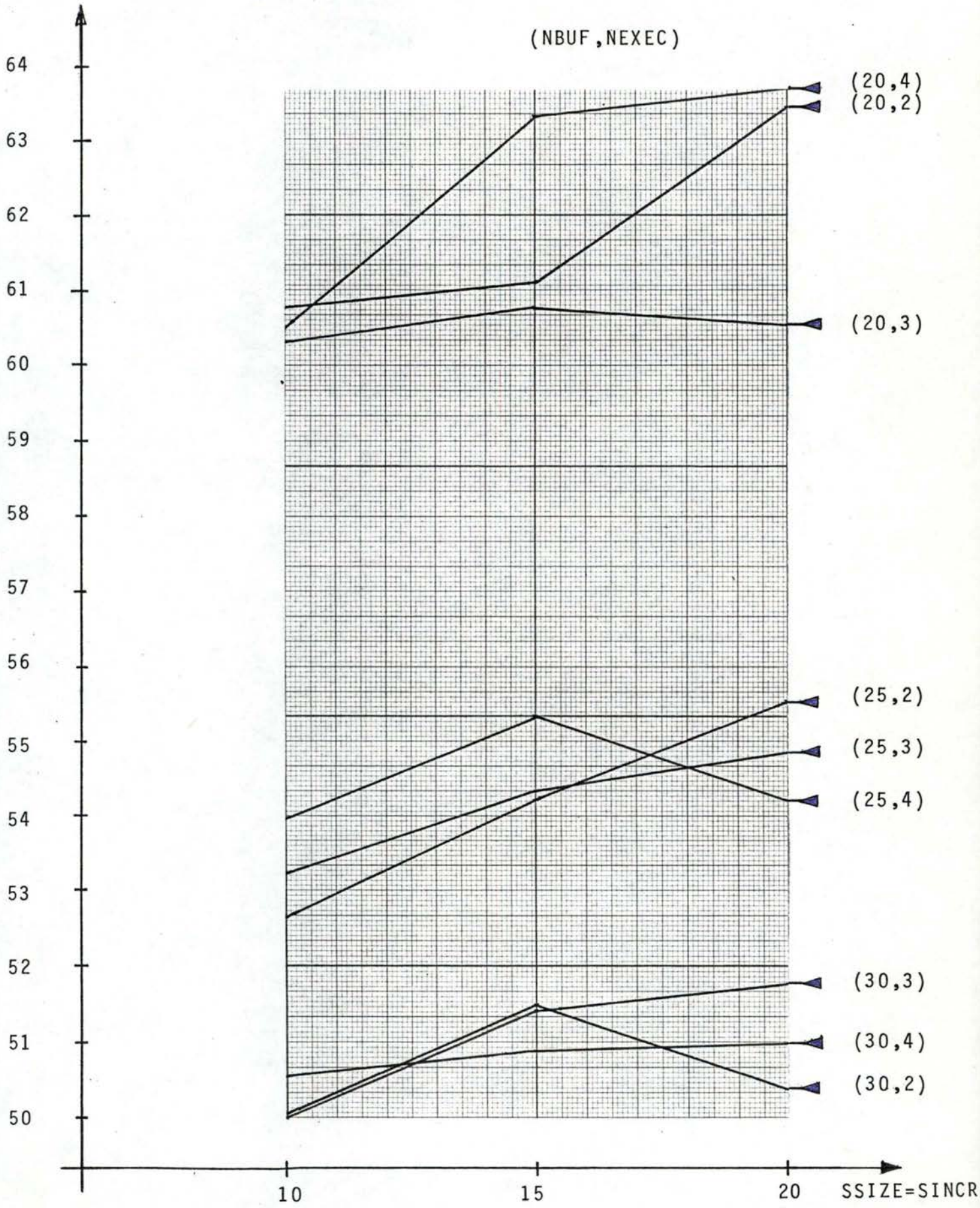
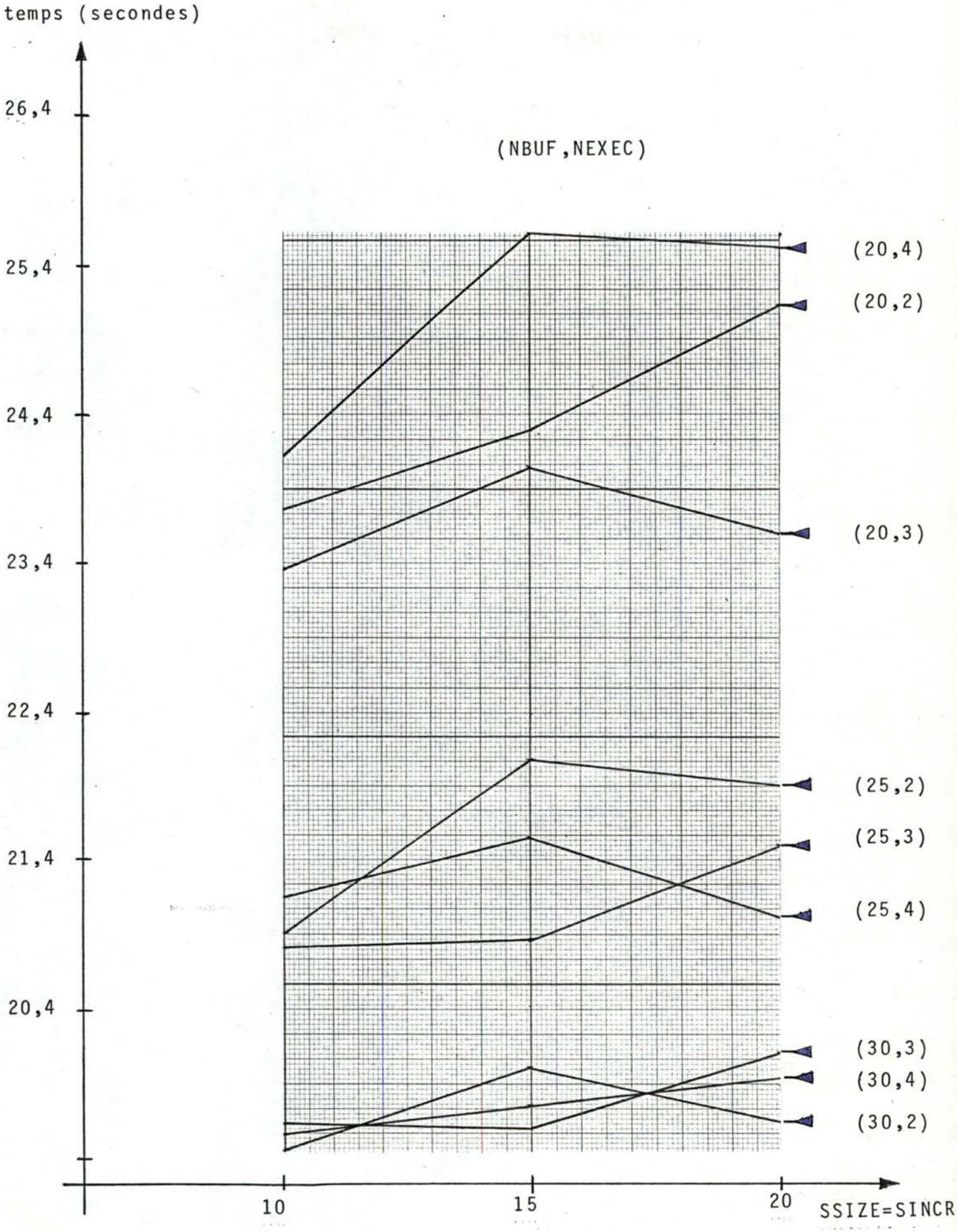


Fig. VI.9 Temps moyen en fonction de SSIZE=SINCR



La figure VI.11 donne l'extrait du protocole de l'exécution du programme d'analyse de variance.

```
# anova
How many factors?        3
Analysis of a 3-way crossed design:

How much replications?                    1

Factor  B:        Name:                   NBUF

                  How much levels?        3

Factor  C:        Name:                   NEXEC

                  How much levels?        3

Factor  D:        Name:                   STACK

                  How much levels?        3

On which file are the datas ?             ../../means1
all values read in. starting computing
Computing done.
On which file do you want output?         out.an


anova done.

#
```

Fig. VI.11 Exécution de "anova.c"

VI.4.4.1 Analyse de variance.

Rappelons l'expression formelle du modèle utilisé :

Facteurs : B, C, D

Indices : $1 \leq m \leq M$ pour les niveaux de B
$1 \leq l \leq L$ pour les niveaux de C
$1 \leq k \leq K$ pour les niveaux de D
$1 \leq n \leq N$ pour les répétitions

Modèle :

$$\underline{x}_{m,l,k,n} = A + B_m + C_l + D_k + BC_{m,l} + BD_{m,k} + CD_{l,k} + BCD_{m,l,k} + \underline{\varepsilon}_{m,k,n}$$

Avec les contraintes

$$\sum_{m=1}^{M} B_m = 0 \qquad \sum_{l=1}^{L} C_l = 0 \qquad \sum_{k=1}^{K} D_k = 0$$

$$\sum_{m=1}^{M} BC_{n,l} = 0, \forall l \qquad \sum_{l=1}^{L} BC_{m,l} = 0, \forall m$$

$$\sum_{m=1}^{L} CD_{l,k} = 0, \forall k \qquad \sum_{k=1}^{K} CD_{l,k} = 0, \forall l$$

$$\sum_{m=1}^{M} BD_{m,k} = 0, \forall k \qquad \sum_{k=1}^{K} BD_{m,k} = 0, \forall m$$

Rappelons également la définition des sommes de carrées calculées par "anova.c" (tableau VI.5).

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{n=1}^{N} \underline{x}^2_{m,l,k,n}$	KxLxMxN
B	$\frac{1}{KxLxN} \sum_{m=1}^{M} \underline{x}^2_{m,*,*,*}$	M
C	$\frac{1}{MxKxN} \sum_{l=1}^{L} \underline{x}^2_{*,L,*,*}$	L
D	$\frac{1}{MxLxN} \sum_{k=1}^{K} \underline{x}^2_{*,*,k,*}$	K
BC	$\frac{1}{KxN} \sum_{m=1}^{M} \sum_{l=1}^{L} \underline{x}^2_{m,l,**}$	MxL
BD	$\frac{1}{LxN} \sum_{m=1}^{M} \sum_{k=1}^{K} \underline{x}^2_{m,*,k,*}$	MxK
CD	$\frac{1}{MxN} \sum_{l=1}^{L} \sum_{k=1}^{K} \underline{x}^2_{*,l,k,*}$	LxK
BCD	$\frac{1}{N} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \underline{x}^2_{m,l,k,*}$	MxLxK
Cellules	$\frac{1}{MxLxKxN} \underline{x}^2_{*,*,*,*}$	1

Tableau VI.5 Table d'analyse de variance modèle à 3 facteurs.

Pour notre exemple,

$$M = L = K = 3$$

Nous avons signalé que N = 1. Il est facile de voir que la S.C. Totale et la S.C. par cellules sont identiques. Nous définissons donc en outre la S.C. pour le modèle additif qui ne tient pas compte de l'interaction d'ordre 3 (Tableau VI.6).

Effet		Degrés de liberté
Additif	$\frac{1}{KxN} \sum_{m=1}^{L} \sum_{l=1}^{L} \underline{x}^2_{m,l,*,*}$	MxL
	$+ \frac{1}{LxN} \sum_{m=1}^{M} \sum_{k=1}^{K} \underline{x}^2_{m,*,k,*}$	+ MxK
	$+ \frac{1}{MxN} \sum_{l=1}^{L} \sum_{k=1}^{K} \underline{x}^2_{*,l,k,*}$	+ LxK
	$- \frac{1}{KxLxN} \sum_{m=1}^{M} \underline{x}^2_{m,*,*,*}$	- M
	$- \frac{1}{MxKxN} \sum_{l=1}^{L} \underline{x}^2_{*,l,*,*}$	- L
	$- \frac{1}{MxLxN} \sum_{k=1}^{K} \underline{x}^2_{*,*,k,*}$	- K
	$+ \frac{1}{MxLxKxN} \underline{x}^2_{*,*,*,*}$	+ 1

Tableau VI.6 S.C. pour le modèle additif sans les interactions d'ordre 3.

Le tableau VI.7 donne les résultats des calculs des S.C. ainsi définies :

Effet	Somme des carrés	Degrés de liberté	S.C. des erreurs
Total	14179.339	27	0.000
ADD	14178.025	19	1.314
CELL	14070.947	1	108.392
B	14171.597	3	7.742
C	14072.330	3	107.009
BC	14175.160	9	4.179
D	14073.049	3	106.290
BD	14174.152	9	5.187
CD	14074.742	9	104.597
BCD	14179.339	27	0.000

Tableau VI.7 Table d'analyse de variance.

Dans notre cas, les épreuves d'hypothèse se feront par rapport au modèle additif.

Ainsi, si on veut éprouver l'effet B, c'est-à-dire le paramètre NBUF, on calculera

$$\underline{F} = \frac{(14178.025 - 14171.597)/(19-3)}{1.314/8} = 2.446$$

Ce quotient est une variable aléatoire de distribution F à 16 et 8 degrés de liberté. Les tables de cette distribution fournissent le quantile

$$Q_F\ (0.99\ ;\ 16,8) = 5.48$$

comme $\underline{F} < Q_F\ (0.99\ ;\ 16,8)$

on ne peut pas rejeter l'hypothèse sans épreuve et l'effet principal B doit être considéré comme significatif.

On peut calculer les différents quotients et on trouve Effets significatifs : B, BC, BD,
les autres effets n'étant pas significatifs.

On voit donc que le paramètre NBUF joue un rôle très important dans la variation du temps de réponse. Il en est de même pour les interactions d'ordre 2 de NBUF avec NEXEC et STACK.

A ce niveau, on peut conclure en considérant NBUF comme responsable de l'amélioration des performances, puisque l'effet qu'il induit est significatif même s'il est combiné avec un des deux autres facteurs.

Il reste à donner à cette conclusion une explication en considérant les mécanismes où ces facteurs interviennent.

VI.4.4.2 Conclusions des mesures.

Lorsque des mesures de performance sont faites sur la base de variations de paramètres-systèmes, il faut, pour interpréter parfaitement les résultats, une connaissance remarquable de la signification de ces paramètres. L'acquérir c'est en général maîtriser toutes les notions, même les plus infimes, du système.

Bien qu'ayant étudié UNIX, nous ne pouvons prétendre en connaître tous les mécanismes. Aussi il se peut que nous omettions certains détails dans l'interprétation des résultats que nous présentons.

D'abord nous montrons les relations entre des performances enregistrées

a. NBUF (Fig. VI.6, VI.7).

Les tampons pour blocs, de par leur définition, occupent une place mémoire assez importance car leur fonction principale est d'assurer le transit des informations entre la mémoire principale et le disque (et vice-versa), en particulier lors du chargement ou du déchargement de programmes en mémoire.

Leur nombre règle donc le débit des transferts d'informations entre ces deux ressources. L'augmenter revient à assurer un meilleur écoulement.

Cependant ils occupent une place en mémoire principale. Plus ils sont nombreux, moins ils y laissent de l'espace pour d'autres usages.

La nette amélioration constatée lorsqu'ils augmentent, montre l'importance du débit entre le disque et la mémoire principale. Plus il est facilité, mieux c'est !

Cependant, les résultats s'améliorent plus lorsqu'on passe de 20 à 25 tampons que lorsqu'on passe de 25 à 30. Il est pratiquement certain que si on continue à accroître leur nombre, la différence de performances deviendrait de plus en plus faible et finirait par montrer une dégradation de celles-ci. En effet de cette manière, la zone des tampons laisserait de moins en moins de place en mémoire centrale pour les autres usages, notamment pour l'exécution des processus.

Cependant il est bon de noter que les tampons occupent une place dans le code UNIX et que ce code ne peut s'étaler sur plus de 48 KBytes en mémoire.

b) SSIZE=SINCR (Fig. VI.8, VI.9).

Dans la version standard de UNIX, l'unité de pile utilisateur est de 20 (x64 Bytes). Tout nouveau processus reçoit une zone de pile de 1 unité. Si cette place n'est utilisée au maximum qu'à 50%, il y a 640 Bytes de mémoire tantôt principale, tantôt auxiliaire (sur le disque), qui ne sont pas utilisés, et ce, pendant toute l'existence de ce processus. Par contre, si celui-ci a besoin à un certain moment de plus d'espace pour cette zone, il lui est accordé 1 unité supplémentaire de pile. Le processus la conserve aussi jusqu'à la fin de sa vie. La zone de pile peut augmenter mais ne diminue jamais.

En calculant au plus juste la zone de pile, c'est-à-dire en amenant l'unité le plus près possible de l'espace occupé par la pile, on économise sûrement de la mémoire.

Ce gain, pris pour tous les processus, peut conduire à stocker plus de segments de texte et de donnée en mémoire principale et à diminuer ainsi le nombre de transferts de la mémoire principale vers la mémoire auxiliaire (et vice versa). De plus, le temps même de transfert diminue puisqu'il n'y a presque plus de place inutile.

D'un autre côté, une petite unité risque d'amener un nombre plus élevé de demandes d'accroissement de la zone de pile, c'est-à-dire, un travail supplémentaire et donc une perte de temps.

Lorsque l'unité passe de 20 à 15, le résultat est mitigé. Parfois les temps de réponse diminuent, parfois ils augmentent. L'économie mémoire n'est sans doute pas suffisante pour vraiment, en tout cas, surpasser ni même compenser le travail des demandes supplémentaires d'accroissement de pile.

Mais lorsqu'elle passe de 20 à 10, l'indication est claire : les performances en temps sont meilleures. Le gain mémoire est plus fort que le travail supplémentaire.

Ces explications conduisent à conclure que le travail d'un transfert est plus important que celui d'un

accroissement de pile. On arrive à la même conséquence par une étude du code de UNIX.

Il est presque sûr qu'une unité supérieure à 20 dégrade les performances. Mais il n'est pas évident qu'une unité inférieure à 10 les améliore car il n'y a peut-être plus de gain mémoire à obtenir.

c. NEXEC (Fig. VI.4, VI.5).

Ce paramètre détermine le nombre maximum d'exécutions simultanées de la primitive "exec". Presque toutes les commandes y font appel.

Il n'est guère aisé d'expliquer "exec" sans tomber dans une multitude de détails et de notions de UNIX.

En bref, quand "exec" est appelé, il sauve dans un tampon toute l'information nécessaire pour reconstruire le processus transformé (cfr III.2.1 Les primitives). Il libère la place mémoire occupée par le segment de texte et les zones utilisateur de donnée et de pile du processus. Ensuite avec notamment le contenu du tampon, il reconstruit les zones utilisateur de donnée et de pile pour le nouveau processus et il le lie à un nouveau segment de texte. Cependant lors de la reconstruction, s'il n'y a plus de mémoire principale en suffisance, le vieux processus est déchargé vers la mémoire auxiliaire en attendant la place nécessaire pour se reconstruire en le nouveau processus.

Si on augmente NEXEC, plus de processus peuvent être traités simultanément et le temps d'attente pour un de ceux-ci ne peut que diminuer. Mais ainsi il y a plus de demandes de réservations de place en mémoire principale et le nombre de transfert, ainsi que le temps d'attente, en mémoire auxiliaire, est enclin à augmenter.

La tendance générale des résultats montre qu'un accroissement de NEXEC de 3 à 4, fournit apparemment de moins bonnes performances, surtout avec NBUF=20. S'il y a plus de processus servis, ils mettent plus de temps à être traités

(il y a plus de transferts notamment).

Diminuer NEXEC, c'est servir moins de processus mais aussi soulager la gestion de la mémoire.

Le passage de 3 à 2 ne donne aucune indication générale.

Les résultats semblent dépendre fortement des valeurs des autres paramètres.

d. Interprétation globale.

On s'aperçoit que le facteur essentiel dans ces mesures est la mémoire.

Plus elle est occupée, plus les transferts de la mémoire principale à la mémoire auxiliaire (et vice versa) sont importants.

Lorsque ces transferts s'exécutent rapidement, l'encombrement de la mémoire principale n'est plus si important. En effet, on peut la décharger puis la recharger sans perdre énormément de temps.

Mais si le débit permis est faible, la gestion de la mémoire principale prend une certaine valeur qu'il ne faut pas négliger.

La vitesse de transfert dépend de NBUF. L'allocation de la mémoire par processus dépend de SSIZE=SINCE et de NEXEC. Cependant NEXEC intervient à d'autres endroits du système (modification du descripteur d'un processus, appel à des tampons, ...).

Si on donne une valeur suffisante (30) à NBUF, les valeurs des autres paramètres importent peu (dans nos mesures). Par contre, si elle n'est pas assez grande, les autres paramètres jouent un certain rôle (principalement SSIZE=SINCE).

e. Conclusion.

Nos mesures ont essentiellement porté sur la mémoire et tendent à montrer la haute valeur de la mémoire principale en tant que ressource.
Si elle était plus importante ou si une meilleure gestion pouvait être trouvée, nous sommes convaincus que même le rôle de NBUF risque de diminuer dans nos mesures.

CONCLUSION

Nous avons mis au point un procédé de mesure de performance pour le système d'exploitation UNIX.

Par un exemple, nous avons montré que son utilisation est acceptable.

Pour mener à bien ce travail, nous avons dû imposer certaines restrictions qui pourraient faire l'objet d'extensions futures.

Tout d'abord, une étude fondamentale plus poussée pourrait amener un expérimentateur à définir d'autres algorithmes que ceux utilisés dans la version originale de UNIX.

On pourrait dès lors envisager de quantifier des choix d'algorithmes et mesurer leur importance relative.

D'autre part, l'utilisation de plans d'expériences complets conduit, nous l'avons vu, à un nombre élevé de mesures.

Il serait souhaitable d'étendre la procédure à des plans incomplets. Ceux-ci, bien que réduisant les conclusions possibles, peuvent, s'ils sont judicieusement utilisés, faire épargner beaucoup de temps.

Les mesures que nous avons exécutées ont porté sur un modèle général de la future charge de S3. Elle serait éventuellement à revoir ou à détailler lorsque S3 sera installé et sa charge connue avec précision.

BIBLIOGRAPHIE

(1) AMERICAN NATIONAL BUREAU OF STANDARDS : "Guidelines of Interactive Computer Service Response Time and Turnaround Time". Federal Information Processing Standards, Pub 57, August 1978.

(2) BAHR D. (Lufthansa, German Airlines) : "Computer Performance : Principles and Techniques". National Bureau of Standards, U.S. Dept. of Commerce, pub. 406, Computer Performance Evaluation, NBS/ ACM Workshop, 1973.

(3) BOI L., MARTIN R., BOURRET P., MICHEL P., CROS P., TREPOS R., GENTHON P., ROUSSELOT J.-Y. : "Mesure des Systèmes Informatiques", Cepadues Editions, Toulouse 1978 pour Ecole Nationale Supérieure de l'aéronautique et de l'espace.

(4) BOURNE J.C. (University of Texas) : "Performance Factors for University Computer Facilities". National Bureau of Standards, U.S. Dept. of Commerce, pub 406, Computer Performance Evaluation, NBS/ACM Workshop 1973.

(5) BOURNE S.R. : "The UNIX Shell", The Bell System Technical Journal, July/August 1978, Vol. 57, N°6, Part 2, The American Telephone and Telegraph Cy.

(6) COLOURIS G. : "UNIX : Basic Principles", Rapport de Travail Queen Mary College, University of London,

(7) DAVIES O.L. : "Design and Analysis of Industrial Experiments", Oliver and Boyd for ICI Ltd, 1967.

(8) DE RIVET Ph. : "Introduction à la métrologie des Systèmes Informatiques, Notes de cours, F.U.N. 1979.

(9) DIGITAL EQUIPMENT CORPORATION :
"PDP 11/45 Processor Handbook",
"PDP 11 Peripherals Handbook".

(10) FACULTES UNIVERSITAIRES DE NAMUR, Commission Plan-Calcul : "Benchmarks pour le choix des nouveaux moyens informatiques des facultés".

(11) JOHNSON N.L., LEONE F.C. : "Statistics and Experimental Design in Engineering and the Physical Sciences", Vol II, John Wiley & Sons, New York, USA, 1964.

(12) JOHNSON S.C., RITCHIE D.M. : "Portability of C Programs and the UNIX System", the Bell System Technical Journal, July/August 1978, Vol 57, n° 6, Part 2, The American Telephone and Telegraph Cy.

(13) KERNIGHAM B.W. : "Unix For Beginners", Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(14) KERNIGHAM B.W. : "A Tutorial Introduction to the UNIX Text Editor", Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(15) KERNIGHAM B.W. : "The C Programming Language". Prentice Hall Saftware Series, New Jersey, 1978.

(16) KERNIGHAM B.W., RITCHIE : "Programming in C - A Tutorial", "C Reference Manual", Bell Laboratories, Murray Hill, New Jersey, USA, 1978.

(17) LESK M.E. : "The Portable C Library" (on UNIX), Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA 1975.

(18) LIONS J. : "A Commentary on the UNIX Operating System - Level 6". University of New South Wales, Dept. of Computer Science, UK, 1977.

(19) RITCHIE D.M. : "The UNIX I/O System", Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(20) RITCHIE D.M. : "UNIX Assembler Reference Manual", Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(21) RITCHIE D.M. : "On the Security of UNIX", Documents for Use with the UNIX time Sharing System, Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(22) RITCHIE D.M., THOMPSON K.D. : "The UNIX Time Sharing System", The Bell System Technical Journal, July/August 1978, Vol. 57, N°6, Part 2, The American Telephone and Telegraph Cy.

(23) RITCHIE D.M., JOHNSON S.C., LESK M.E., KERNIGHAM B.W. : "The C Programming Language", The Bell System Technical Journal, July/August 1978, Vol. 57, N°6, Part 2, The American Telephone and Telegraph Cy.

(24) ROSE C.A. : "Performance Measurement", Computing Surveys", Vol. 10, N°3, September 1978.

(25) ROWSON J. : "An introduction to UNIX", Notes de cours, Queen Mary College, University of London, UK, 1978.

(26) THOMPSON K.D. : "UNIX Implementation", The Bell System Technical Journal, July/August 1978, Vol. 57, N°6, Part 2, The American Telephone and Telegraph Cy.

(27) THOMPSON K.D., RITCHIE D.M. : "The UNIX Programmers Manual", Bell Laboratories, Murray Hill, New Jersey, USA, 1975.

(28) TURNER R., LEVY H. (Digital Equipment Corporation) : "Performance Evaluation of IAS on the PDP" Symposium on Computer Performance Modelling, Measurement and Evaluation, ACM/Sigmetric, IFIP Working Group, 1976.

(29) WARNER C.D. (Test data Systems Corporation) : Measurement Tools", National Bureau of Standards US. Dept. of Commerce, pub 406, Computer Performance Evaluation, NBS/ACM Workshop 1973.

(30) Western Electric Corporation, UNIX Operating System Source Code level 6.Ø (Licence).

—

ANNEXES

Annexe A.1

Définition des sommes de carrés relatives à des plans croisés complets de 1 à 5 facteurs

---

Remarque : Nous ne considérons ici que les modèles pour lesquels toutes les cellules ont le même nombre d'observations, soit N.

A.1.1 Plan à 1 facteur

Facteur B

$1 \leqslant m \leqslant M \; ; \; 1 \leqslant n \leqslant N$

Modèle $\underline{x}_{m,n} = A + B_m + \underline{\varepsilon}_{m,n}$

Contrainte $\sum_{m=1}^{M} B_m = 0$

Table d'analyse de variance

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{m=1}^{M} \sum_{n=1}^{N} \underline{x}^2_{m,n}$	MN
B	$\frac{1}{N} \sum_{m=1}^{M} \underline{x}^2_{m,*}$	M
Cellules	$\frac{1}{MN} \underline{x}^2_{*,*}$	1

A.1.2 Plan à 2 facteurs

Facteurs B,C

$1 \leqslant m \leqslant M \; ; \; 1 \leqslant l \leqslant L \; ; \; 1 \leqslant n \leqslant N$

Modèle $\underline{x}_{m,l,n} = A + B_m + C_l + BC_{m,l} + \underline{\varepsilon}_{m,l,n}$

Contraintes $\sum_{m=1}^{M} B_m = \sum_{l=1}^{L} C_l = 0$;

$\sum_{n=1}^{M} BC_{m,l} = 0 \; \forall l \; ; \; \sum_{l=1}^{L} BC_{m,l} = 0 \; \forall m$

Table d'analyse de variance

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{n=1}^{N} \underline{x}^2_{m,l,n}$	MLN
B	$\frac{1}{LN} \sum_{n=1}^{M} \underline{x}^2_{m,*,*}$	M
C	$\frac{1}{MN} \sum_{l=1}^{L} \underline{x}^2_{*,l,*}$	L
BC	$\frac{1}{N} \sum_{m=1}^{M} \sum_{l=1}^{} \underline{x}^2_{m,l,*}$	ML
Cellules	$\frac{1}{MLN} \underline{x}^2_{*,*,*}$	1
Additif	SC.B + SC.C - SC.Cellules	M+L+1

A.1.3 Plan à 3 facteurs

Facteurs B,C,D

$1 \leqslant m \leqslant M \; ; \; 1 \leqslant l \leqslant L \; ; \; 1 \leqslant k \leqslant K \; ; \; 1 \leqslant n \leqslant N$

Modèle

$$\underline{x}_{m,l,k,n} = 1 + B_m + C_l + D_k + BC_{m,l} + BD_{m,k} + CD_{l,k} + BCD_{m,l,k} + \underline{\varepsilon}_{m,l,k,n}$$

Contraintes

$$\sum_{m=1}^{M} B_m = \sum_{l=1}^{L} C_l = \sum_{k=1}^{K} D_k = 0 ;$$

$$\sum_{m=1}^{M} BC_{m,l} = 0 \; \forall l \; ; \; \sum_{l=1}^{L} BC_{m,l} = 0 \; \forall m \; ;$$

$$\sum_{m=1}^{L} BD_{m,k} = 0 \; \forall k \; ; \; \sum_{k=1}^{K} BD_{m,k} = 0 \; \forall m \; ;$$

$$\sum_{l=1}^{L} CD_{l,k} = 0 \; \forall k \; ; \; \sum_{k=1}^{K} CD_{l,k} = 0 \; \forall l .$$

Table d'analyse de variance

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{m=1}^{M}\sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{n=1}^{N} \underline{x}^2_{m,l,k,n}$	MLKN
B	$\frac{1}{LKN}\sum_{m=1}^{M} \underline{x}^2_{m,*,*,*}$	M
C	$\frac{1}{MKN}\sum_{l=1}^{L} \underline{x}^2_{*,l,*,*}$	L
D	$\frac{1}{MLN}\sum_{k=1}^{K} \underline{x}^2_{*,*,k,*}$	K
BC	$\frac{1}{KN}\sum_{m=1}^{M}\sum_{l=1}^{L} \underline{x}^2_{m,l,*,*}$	ML
BD	$\frac{1}{LN}\sum_{m=1}^{M}\sum_{k=1}^{K} \underline{x}^2_{m,*,k,*}$	MK
CD	$\frac{1}{MN}\sum_{l=1}^{L}\sum_{k=1}^{K} \underline{x}^2_{*,l,k,*}$	LK
BCD	$\frac{1}{N}\sum_{m=1}^{M}\sum_{l=1}^{L}\sum_{k=1}^{K} \underline{x}^2_{m,l,k,*}$	MLK
Cellules	$\frac{1}{MLKN} \underline{x}^2_{*,*,*,*}$	1

Additif	SC.BC + SS.BD + SS.CD - SS.B - SS.C - SS.D + SS.Cellules	ML + MK + LK - M - L - K + 1

A.1.4. plan à 4 facteurs

Facteurs B,C,D,E

$1 \leqslant m \leqslant M$; $1 \leqslant l \leqslant L$; $1 \leqslant k \leqslant K$; $1 \leqslant j \leqslant J$;

$1 \leqslant n \leqslant N$

Modèle

$$\underline{x}_{m,l,k,j,n} = A + B_m + C_l + D_k + E_j$$
$$+ BC_{m,l} + BD_{m,k} + BE_{m,j} + CD_{l,k} + CE_{l,j}$$
$$+ DE_{k,j} + BCD_{n,l,k} + BCE_{m,l,j} + BDE_{m,k,j}$$
$$+ CDE_{l,k,j} + BCDE_{n,l,k,j} + \underline{\varepsilon}_{m,l,k,j,n}$$

Contraintes

$$\sum_{n=1}^{M} B_m = \sum_{l=1}^{L} C_l = \sum_{k=1}^{K} D_k = \sum_{j=1}^{J} E_j = 0 \ ;$$

$$\sum_{m=1}^{M} BC_{m,l} = 0 \ \forall l \ ; \ \sum_{l=1}^{L} BC_{m,l} = 0 \ \forall m \ ;$$

$$\sum_{m=1}^{M} BD_{m,k} = 0 \ \forall k \ ; \ \sum_{k=1}^{K} BD_{m,k} = 0 \ \forall m \ ;$$

$$\sum_{m=1}^{M} BE_{m,j} = 0 \ \forall j \ ; \ \sum_{j=1}^{J} BE_{m,j} = 0 \ \forall m \ ;$$

$$\sum_{l=1}^{L} CD_{l,k} = 0 \ \forall k \ ; \ \sum_{k=1}^{K} CD_{l,k} = 0 \ \forall l \ ;$$

$$\sum_{l=1}^{L} CE_{l,j} = 0 \ \forall j \ ; \ \sum_{j=1}^{J} CE_{l,j} = 0 \ \forall l \ ;$$

$$\sum_{k=1}^{K} DE_{k,j} = \forall j \ ; \ \sum_{j=1}^{J} DE_{k,j} = 0 \ \forall k \ ;$$

$$\sum_{n=1}^{M} BCD_{m,l,k} = 0 \ \forall l,k \ ; \ \sum_{m=1}^{M} BDE_{m,k,j} = 0 \ \forall k,j \ ;$$

$$\sum_{l=1}^{L} BCD_{m,l,k} = 0 \ \forall m,k \ ; \ \sum_{k=1}^{K} BDE_{m,k,j} = 0 \ \forall m,j \ ;$$

$$\sum_{k=1}^{K} BCD_{m,l,k} = 0 \ \forall m,l \ ; \ \sum_{j=1}^{J} BDE_{m,k,j} = 0 \ \forall m,k \ ;$$

$$\sum_{m=1}^{M} BCE_{m,l,j} = 0 \ \forall l,j \ ; \ \sum_{l=1}^{L} CDE_{l,k,j} = 0 \ \forall k,j \ ;$$

$$\sum_{l=1}^{L} BCE_{m,l,j} = 0 \ \forall m,j \ ; \ \sum_{k=1}^{K} CDE_{l,k,j} = 0 \ \forall l,j \ ;$$

$$\sum_{j=1}^{J} BCE_{m,l,j} = 0 \ \forall m,l \ ; \ \sum_{j=1}^{J} CDE_{l,k,j} = 0 \ \forall l,k \ .$$

Table d'analyse de variance

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{m=1}^{M}\sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{j=1}^{J}\sum_{n=1}^{N} \underline{x}^2_{m,l,k,n}$	MLKJN
B	$\frac{1}{KJN}\sum_{m=1}^{M} \underline{x}^2_{m,*,*,*,*}$	M
C	$\frac{1}{MKJN}\sum_{l=1}^{L} \underline{x}^2_{*,l,*,*,*}$	L
D	$\frac{1}{MLJN}\sum_{k=1}^{K} \underline{x}^2_{*,*,k,*,*}$	K
E	$\frac{1}{MLKN}\sum_{j=1}^{J} \underline{x}^2_{*,*,*,j,*}$	J
BC	$\frac{1}{KJN}\sum_{m=1}^{M}\sum_{l=1}^{L} \underline{x}^2_{m,l,*,*,*}$	ML
BD	$\frac{1}{LJN}\sum_{m=1}^{M}\sum_{k=1}^{K} \underline{x}^2_{m,*,k,*,*}$	MK
BE	$\frac{1}{LKN}\sum_{m=1}^{M}\sum_{j=1}^{J} \underline{x}^2_{m,*,*,j,*}$	MJ
CD	$\frac{1}{MJN}\sum_{l=1}^{L}\sum_{k=1}^{K} \underline{x}^2_{*,l,k,*,*}$	LK
CE	$\frac{1}{MKN}\sum_{l=1}^{L}\sum_{j=1}^{L} \underline{x}^2_{*,l,*,j,*}$	LJ
DE	$\frac{1}{MLN}\sum_{k=1}^{K}\sum_{j=1}^{J} \underline{x}^2_{*,*,k,j,*}$	KJ
BCD	$\frac{1}{JN}\sum_{m=1}^{M}\sum_{l=1}^{L}\sum_{k=1}^{K} \underline{x}^2_{m,l,k,*,*}$	MLK

BCE	$\frac{1}{KN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{j=1}^{J} \underline{x}^2_{m,l,*,j,*}$	MLJ
BDE	$\frac{1}{LN} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{m,*,k,J,*}$	MKJ
CDE	$\frac{1}{MN} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{*,l,k,j,*}$	LKJ
BCDE	$\frac{1}{N} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{m,l,k,j,*}$	MLKJ
Cellules	$\frac{1}{MLKJN} \underline{x}^2_{*,*,*,*,*}$	1
Additif	SC.BCD + SS.BCE + SS.BDE + SS.CDE - SS.BC - SS.BD - SS.BE - SS.CD - SS.CE - SS.DE + SS.B + SS.C + SS.D + SS.E - SS.Cellules	MLK + MLJ + MKJ + LKJ - ML - MK - MJ - LK - LJ - KJ + M + L + K + J - 1

A.1.5 Plan à 5 facteurs

Facteurs B,C,D,E,F

$1 \leqslant m \leqslant M$; $1 \leqslant l \leqslant L$; $1 \leqslant k \leqslant K$; $1 \leqslant j \leqslant J$;
$1 \leqslant i \leqslant I$; $1 \leqslant n \leqslant N$.

Modèle :

$$\underline{x}_{m,l,k,j,i,n} = A + B_m + C_l + D_k + E_j + F_i$$

$$+ BC_{m,l} + BD_{m,k} + BE_{m,j} + BF_{m,i}$$

$$+ CD_{l,k} + CE_{l,j} + CF_{l,i} + DE_{k,j}$$

$$+ DF_{k,i} + EF_{j,i}$$

$$+ BCD_{m,l,k} + BCE_{m,l,j} + BCF_{m,l,i}$$

$$+ BDE_{m,k,j} + BDF_{m,k,i} + BEF_{m,j,i}$$

$$+ CDE_{l,k,j} + CDF_{l,k,i} + CEF_{l,j,i}$$

$$+ DEF_{k,j,i}$$

$$+ BCDE_{m,l,k,j} + BCDF_{m,l,k,i} + BCEF_{m,l,j,i}$$

$$+ BDEF_{m,k,j,i} + CDEF_{l,k,j,i}$$

$$+ BCDEF_{m,l,k,j,i}$$

$$+ \underline{\varepsilon}_{m,l,k,j,i,n}$$

Contraintes

$$\sum_{n=1}^{M} B_m = \sum_{l=1}^{L} C_l = \sum_{k=1}^{K} D_K = \sum_{j=1}^{J} E_j = \sum_{i=1}^{I} F_i = 0 \ ;$$

$$\sum_{n=1}^{M} BC_{m,l} = 0 \ \forall l \ ; \ \sum_{l=1}^{L} BC_{m,l} = 0 \ \forall m \ ;$$

$$\sum_{m=1}^{M} BD_{m,k} = 0 \ \forall k \ ; \ \sum_{k=1}^{K} BD_{m,k} = 0 \ \forall m \ ;$$

$$\sum_{m=1}^{M} BE_{m,j} = 0 \ \forall j \ ; \ \sum_{j=1}^{J} BE_{m,j} = 0 \ \forall m \ ;$$

$$\sum_{m=1}^{M} BF_{m,i} = 0 \ \forall i \ ; \ \sum_{i=1}^{I} BF_{m,i} = 0 \ \forall m \ ;$$

$$\sum_{l=1}^{L} CD_{l,k} = 0 \ \forall k \ ; \ \sum_{k=1}^{K} CD_{l,k} = 0 \ \forall l \ ;$$

$$\sum_{l=1}^{L} CE_{l,j} = 0 \ \forall j \ ; \ \sum_{j=1}^{J} CE_{l,j} = 0 \ \forall l \ ;$$

$$\sum_{l=1}^{L} CF_{l,i} = 0 \ \forall i \ ; \ \sum_{i=1}^{I} CF_{l,i} = 0 \ \forall l \ ;$$

$$\sum_{k=1}^{K} DE_{k,j} = 0 \ \forall j \ ; \ \sum_{j=1}^{J} DE_{k,j} = 0 \ \forall k \ ;$$

$$\sum_{k=1}^{K} DF_{k,i} = 0 \ \forall i \ ; \ \sum_{i=o}^{I} DF_{k,i} = 0 \ \forall k \ ;$$

$$\sum_{j=1}^{J} EF_{j,i} = 0 \ \forall i \ ; \ \sum_{i=o}^{I} EF_{j,i} = 0 \ \forall j \ ;$$

$$\sum_{n=1}^{M} BCD_{m,l,k} = 0 \ \forall l,k \ ; \ \sum_{l=1}^{L} BCD_{m,l,k} = 0 \ \forall m,k \ ;$$

$$\sum_{k=1}^{K} BCD_{m,l,k} = 0 \ \forall m,l \ ; \ \sum_{m=1}^{M} BCE_{m,l,j} = 0 \ \forall l,j \ ;$$

$$\sum_{l=1}^{L} BCE_{m,l,j} = 0 \;\forall m,j \;;\quad \sum_{j=1}^{J} BCE_{m,l,j} = 0 \;\forall m,l \;;$$

$$\sum_{m=1}^{M} BCF_{m,l,i} = 0 \;\forall l,i \;;\quad \sum_{l=1}^{L} BCF_{m,l,i} = 0 \;\forall m,i \;;$$

$$\sum_{i=1}^{I} BOF_{m,l,i} = 0 \;\forall m,l \;;\quad \sum_{m=1}^{M} BDE_{m,k,j} = 0 \;\forall k,j \;;$$

$$\sum_{k=1}^{K} BDE_{m,k,j} = 0 \;\forall m,j \;;\quad \sum_{j=1}^{J} BDE_{m,k,j} = 0 \;\forall m,k \;;$$

$$\sum_{m=1}^{M} BDF_{m,k,i} = 0 \;\forall k,i \;;\quad \sum_{k=1}^{K} BDF_{m,k,i} = 0 \;\forall m,i \;;$$

$$\sum_{i=1}^{I} BDF_{m,k,i} = 0 \;\forall m,k \;;\quad \sum_{m=1}^{M} BEF_{m,j,i} = 0 \;\forall j,i \;;$$

$$\sum_{j=1}^{J} BEF_{m,j,i} = 0 \;\forall m,i \;;\quad \sum_{i=1}^{I} BEF_{m,j,i} = 0 \;\forall m,j \;;$$

$$\sum_{l=1}^{L} CDE_{l,k,j} = 0 \;\forall k,j \;;\quad \sum_{k=1}^{K} CDE_{l,k,j} = 0 \;\forall l,j \;;$$

$$\sum_{j=1}^{J} CDE_{l,k,j} = 0 \;\forall l,k \;;\quad \sum_{l=1}^{L} CDF_{l,k,i} = 0 \;\forall k,i \;;$$

$$\sum_{l=1}^{L} CEF_{l,j,i} = 0 \;\forall j,i \;;\quad \sum_{j=1}^{J} CEF_{l,j,i} = 0 \;\forall l,i \;;$$

$$\sum_{i=1}^{I} CEF_{l,j,i} = 0 \;\forall l,j \;;\quad \sum_{k=1}^{K} DEF_{k,j,i} = 0 \;\forall j,i \;;$$

$$\sum_{j=1}^{J} DEF_{k,j,i} = 0 \;\forall k,i \;;\quad \sum_{i=1}^{I} DEF_{k,j,i} = 0 \;\forall k,j \;;$$

$$\sum_{m=1}^{M} BCDE_{m,l,k,j} = 0 \;\forall l,k,j \;;\quad \sum_{l=1}^{L} BCDE_{m,l,k,j} = 0 \;\forall m,k,j \;;$$

$$\sum_{k=1}^{K} BCDE_{m,l,k,j} = 0 \;\forall m,l,j \;;\quad \sum_{l=1}^{L} BCDE_{m,l,k,j} = 0 \;\forall m,l,k \;;$$

$$\sum_{m=1}^{M} BCDF_{m,l,k,i} = 0 \quad \forall l,k,i \; ; \quad \sum_{l=1}^{L} BCDF_{m,l,k,i} = 0 \quad \forall m,k,i \; ;$$

$$\sum_{k=1}^{K} BCDF_{m,l,k,i} = 0 \quad \forall m,l,i \; ; \quad \sum_{i=1}^{I} BCDF_{m,l,k,i} = 0 \quad \forall m,l,k \; ;$$

$$\sum_{M=1}^{M} BDEF_{m,k,j,i} = 0 \quad \forall k,j,i \; ; \quad \sum_{k=1}^{K} BDEF_{m,k,j,i} = 0 \quad \forall m,j,i \; ;$$

$$\sum_{j=1}^{J} BDEF_{m,k,j,i} = 0 \quad \forall m,k,i \; ; \quad \sum_{i=1}^{I} BDEF_{m,k,j,i} = 0 \quad \forall m,k,j \; ;$$

$$\sum_{l=1}^{L} CDEF_{l,k,j,i} = 0 \quad \forall k,j,i \; ; \quad \sum_{k=1}^{K} CDEF_{l,k,j,i} = 0 \quad \forall l,j,i \; ;$$

$$\sum_{j=1}^{J} CDEF_{l,k,j,i} = 0 \quad \forall l,k,i \; ; \quad \sum_{i=1}^{I} CDEF_{l,k,j,i} = 0 \quad \forall l,k,j \; ;$$

Table d'analyse de variance

Effet	Somme de carrés	Degrés de liberté
Total	$\sum_{n=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{i=1}^{I} \sum_{n=1}^{N} \underline{x}^2_{m,l,k,j,n}$	MLKJIN
B	$\frac{1}{LKJIN} \sum_{m=1}^{M} \underline{x}^2_{m,*,*,*,*,*}$	M
C	$\frac{1}{MKJIN} \sum_{l=1}^{L} \underline{x}^2_{*,l,*,*,*,*}$	L
D	$\frac{1}{MLJIN} \sum_{k=1}^{K} \underline{x}^2_{*,*,k,*,*,*}$	K
E	$\frac{1}{MLKIN} \sum_{j=1}^{J} \underline{x}^2_{*,*,*,j,*,*}$	J
F	$\frac{1}{MLKJN} \sum_{i=1}^{I} \underline{x}^2_{*,*,*,*,i,*}$	I
BC	$\frac{1}{MJIN} \sum_{m=1}^{M} \sum_{l=1}^{L} \underline{x}^2_{m,l,*,*,*,*}$	ML
BD	$\frac{1}{LJIN} \sum_{m=1}^{M} \sum_{k=1}^{K} \underline{x}^2_{m,*,k,*,*,*}$	MK
BE	$\frac{1}{LKIN} \sum_{m=1}^{M} \sum_{j=1}^{J} \underline{x}^2_{m,*,*,j,*,*}$	MJ
BF	$\frac{1}{LKJN} \sum_{m=1}^{M} \sum_{i=1}^{I} \underline{x}^2_{m,*,*,*,i,*}$	MI
CD	$\frac{1}{MJIN} \sum_{l=1}^{L} \sum_{k=1}^{K} \underline{x}^2_{*,l,k,*,*,*}$	LK

CE	$\frac{1}{MKIN} \sum_{l=1}^{L} \sum_{j=1}^{J} \underline{x}^2_{*,l,*,j,*,*}$	LJ
CF	$\frac{1}{MKJN} \sum_{l=1}^{L} \sum_{i=1}^{I} \underline{x}^2_{*,l,*,*,i,*}$	LI
DE	$\frac{1}{MLIN} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{*,*,k,*,*}$	KJ
DF	$\frac{1}{MLJN} \sum_{k=1}^{K} \sum_{i=1}^{I} \underline{x}^2_{*,*,k,*,i,*}$	KI
EF	$\frac{1}{MLKN} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{*,*,*,j,i,*}$	JI
BCD	$\frac{1}{JIN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \underline{x}^2_{m,l,k,*,*,*}$	MLK
BCE	$\frac{1}{NIN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{j=1}^{J} \underline{x}^2_{m,l,*,j,*,*}$	MLJ
BCF	$\frac{1}{KJN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{i=1}^{I} \underline{x}^2_{m,l,*,*,i,*}$	MLI
BDE	$\frac{1}{LIN} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{m,*,k,j,*,*}$	MKJ
BDF	$\frac{1}{LJN} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{i=1}^{I} \underline{x}^2_{m,*,k,*,i,*}$	MKI
BEF	$\frac{1}{LKN} \sum_{m=1}^{M} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{m,*,*,j,i,*}$	MJI
CDE	$\frac{1}{MIN} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{*,l,k,j,*,*}$	LKJ
CDF	$\frac{1}{MJN} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=1}^{I} \underline{x}^2_{*,l,k,*,i,*}$	LKI
CEF	$\frac{1}{MKN} \sum_{l=1}^{L} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{*,l,*,j,i,*}$	LJI

DEF	$\frac{1}{MLN} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{*,*,k,j,i,*}$	KJI
BCDE	$\frac{1}{IN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \underline{x}^2_{m,l,k,j,*,*}$	MLKJ
BCDF	$\frac{1}{JN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=1}^{I} \underline{x}^2_{m,l,k,*,i,*}$	MLKI
BCEF	$\frac{1}{KN} \sum_{m=1}^{M} \sum_{l=1}^{L} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{m,l,*,j,i,*}$	MLJI
BDEF	$\frac{1}{LN} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{m,*,k,j,i,*}$	MKJI
CDEF	$\frac{1}{MN} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{i=1}^{I} \underline{x}^2_{*,l,k,j,i,*}$	LKJI
Cellule	$\frac{1}{MLKJIN} \underline{x}^2_{*,*,*,*,*,*}$	1

Additif	SC.NCDE + SC.NCDF + SC.BCEF	MLKJ+MLKI+MLJI
	+ SC.BDEF + SC.CDEF	+ MKJI + LKJI
	- SC.BCD - SC.BCE - SC.BCF	- MLK-MLJ-MLI
	- SC.BDE - SC.BDF - SC.BEF	- MKJ-MKI-MJI
	- SC.CDE - SC.CDF - SC.CEF	- LKJ-LKI-LJI
	- SC.DEF	- KJI
	+ SC.BC + SC.BD + SC.BE + SC.BF	- ML+MK+MJ+MI
	+ SC.CD + SC.CE + SC.CF + SC.DE	+ LK+LJ+LI+KJ
	+ SC.DF + SC.EF	+ KI
	- SC.B - SC.C - SC.D - SC.E - SC.F	- M-L-K-J-I
	+ SC.Cellules	+ 1

Annexe A.2

Modifications apportées au Système UNIX

```
1       #
2       /*
3        */
4
5       /*
6        * This table is the switch used to transfer
7        * to the appropriate routine for processing a system call.
8        * Each row contains the number of arguments expected
9        * and a pointer to the routine.
10       */
11      int     sysent[]
12      {
13              0, &nullsys,                    /*  0 = indir */
14              0, &rexit,                      /*  1 = exit */
15              0, &fork,                       /*  2 = fork */
16              2, &read,                       /*  3 = read */
17              2, &write,                      /*  4 = write */
18              2, &open,                       /*  5 = open */
19              0, &close,                      /*  6 = close */
20              0, &wait,                       /*  7 = wait */
21              2, &creat,                      /*  8 = creat */
22              2, &link,                       /*  9 = link */
23              1, &unlink,                     /* 10 = unlink */
24              2, &exec,                       /* 11 = exec */
25              1, &chdir,                      /* 12 = chdir */
26              0, &gtime,                      /* 13 = time */
27              3, &mknod,                      /* 14 = mknod */
28              2, &chmod,                      /* 15 = chmod */
29              2, &chown,                      /* 16 = chown */
30              1, &sbreak,                     /* 17 = break */
31              2, &stat,                       /* 18 = stat */
32              2, &seek,                       /* 19 = seek */
33              0, &getpid,                     /* 20 = getpid */
34              3, &smount,                     /* 21 = mount */
35              1, &sumount,                    /* 22 = umount */
36              0, &setuid,                     /* 23 = setuid */
37              0, &getuid,                     /* 24 = getuid */
38              0, &stime,                      /* 25 = stime */
39              3, &ptrace,                     /* 26 = ptrace */
40              0, &nosys,                      /* 27 = x */
41              1, &fstat,                      /* 28 = fstat */
42              0, &nosys,                      /* 29 = x */
43              1, &nullsys,                    /* 30 = smdate; inoperative */
44              1, &stty,                       /* 31 = stty */
45              1, &gtty,                       /* 32 = gtty */
46              0, &nosys,                      /* 33 = x */
47              0, &nice,                       /* 34 = nice */
48              0, &sleep,                      /* 35 = sleep */
49              0, &sync,                       /* 36 = sync */
50              1, &kill,                       /* 37 = kill */
51              0, &getswit,                    /* 38 = switch */
52              0, &nosys,                      /* 39 = x */
53              0, &nosys,                      /* 40 = x */
54              0, &dup,                        /* 41 = dup */
55              0, &pipe,                       /* 42 = pipe */
56              1, &times,                      /* 43 = times */
57              4, &profil,                     /* 44 = prof */
58              0, &nosys,                      /* 45 = tiu */
59              0, &setgid,                     /* 46 = setgid */
60              0, &getgid,                     /* 47 = getgid */
61              2, &ssig,                       /* 48 = sig */
62              0, &reboot,                     /* 49 = reboot, FUN NAMUR */
63              0, &nosys,                      /* 50 = x */
64              0, &skill,                      /* 51 = skill, FUN NAMUR */
65              0, &nosys,                      /* 52 = x */
66              0, &nosys,                      /* 53 = x */
67              0, &nosys,                      /* 54 = x */
68              0, &nosys,                      /* 55 = x */
69              0, &nosys,                      /* 56 = x */
70              0, &nosys,                      /* 57 = x */
71              0, &nosys,                      /* 58 = x */
72              0, &nosys,                      /* 59 = x */
73              0, &nosys,                      /* 60 = x */
74              0, &nosys,                      /* 61 = x */
75              0, &nosys,                      /* 62 = x */
76              0, &nosys                       /* 63 = x */
77      };
78
```

```
261     reboot()
262     {
263             update();
264             switch( cputype ) {
265
266                     case 45 :       aloop45(); break;
267                     case 70 :       aloop70(); break;
268                     default :       break;
269             }
270     }
271
272     skill()
273     {
274                     /* special kill system call --- FUN NAMUR */
275                     /* kill all processes attached to the current tty */
276                     /* but the first one, which is assumed to be the running shell */
277                     /* and the one of the caller */
278
279             register struct proc *p;
280             register int actproc ;
281             register int actty ;
282             int count;
283
284             p = u.u_procp;
285             count = 0;
286             actproc = p->p_pid;
287             actty = p->p_ttyp;
288
289             for( p = &proc[0]; p < &proc[NPROC]; p++)
290                     if(p->p_ttyp == actty) {
291                             count++;
292                             if((count > 2) && (p->p_pid != actproc))
293                                     psignal(p, SIGKIL);
294                     }
295     }
296
```

A.2.3 Gestion des lignes de communication

```
1       #
2       /*
3        * getty -- adapt to terminal speed on dialup, and call login
4        */
5
6       /*
7        * tty flags
8        */
9       #define HUPCL 01
10      #define XTABS   02
11      #define LCASE   04
12      #define ECHO    010
13      #define CRMOD   020
14      #define RAW     040
15      #define ODDP    0100
16      #define EVENP   0200
17      #define ANYP    0300
18
19      /*
20       * Delay algorithms
21       */
22      #define CR1     010000
23      #define CR2     020000
24      #define CR3     030000
25      #define NL1     000400
26      #define NL2     001000
27      #define NL3     001400
28      #define TAB1    002000
29      #define TAB2    004000
30      #define TAB3    006000
31      #define FF1     040000
32
33      #define ERASE   '#'
34      #define KILL    '@'
```

```
35
36      /*
37       * speeds
38       */
39      #define B110    3
40      #define B150    5
41      #define B300    7
42      #define B2400   11
43      #define B9600   13
44
45      #define SIGINT  2
46      #define SIGQIT  3
47
48      struct  sgtty {
49              char    sgispd, sgospd;
50              char    sgerase, sgkill;
51              int     sgflag;
52      } tmode;
53
54      struct  tab {
55              int     tname;          /* this table name */
56              int     nname;          /* successor table name */
57              int     iflags;         /* initial flags */
58              int     fflags;         /* final flags */
59              int     ispeed;         /* input speed */
60              int     ospeed;         /* output speed */
61              char    *message;       /* login message */
62      } itab[] {
63
64      /* table '0'-1-2 300,150,110 */
65
66              '0', 1,
67              ANYP+RAW+NL1+CR1, ANYP+ECHO+CR1,
68              B300, B300,
69              "\n\r\033;\007login: ",
70
71              1, 2,
72              ANYP+RAW+NL1+CR1, EVENP+ECHO+FF1+CR2+TAB1+NL1,
73              B150, B150,
74              "\n\r\033:\006\006\017login: ",
75
76              2, '0',
77              ANYP+RAW+NL1+CR1, ANYP+ECHO+CRMOD+XTABS+LCASE+CR1,
78              B110, B110,
79              "\n\rlogin: ",
80
81      /* table '-' -- Console TTY 110 */
82              '-', '-',
83              ANYP+RAW+NL1+CR1, ANYP+ECHO+CRMOD+XTABS+LCASE+CR1,
84              B110, B110,
85              "\n\rlogin: ",
86
87      /* table '1' -- 150 */
88              '1', '1',
89              ANYP+RAW+NL1+CR1, EVENP+ECHO+FF1+CR2+TAB1+NL1,
90              B150, B150,
91              "\n\r\033:\006\006\017login: ",
92
93      /* table '2' -- 2400 */
94              '2', '2',
95              ANYP+RAW+NL1+CR1, ANYP+XTABS+ECHO+CRMOD+FF1,
96              B2400, B2400,
97              "\n\r\033;login: ",
98      };
99
100     #define NITAB   sizeof itab/sizeof itab[0]
101
102     char    name[16];
103     int     crmod;
104     int     upper;
105     int     lower;
106
107     main(argc, argv)
108     char **argv;
109     {
110             register struct tab *tabp;
111             register tname;
112
113     /*
114             signal(SIGINT, 1);
115             signal(SIGQIT, 0);
116     */
117             tname = '0';
118             if (argc > 1)
119                     tname = *argv[1];
```

```
120             switch(tname){
121                     case 'a' : /* 'user' autolog at 110 bauds */
122                     case 'b' : /* 'user' autolog at 300 bauds */
123                     case 'c' : /* 'user' autolog at 2400 bauds */
124                     case 'd' : /* 'guest' autolog at 110 bauds */
125                     case 'e' : /* 'guest' autolog at 2400 bauds */
126                     case 'x' : /* 'root' autolog at 110 bauds */
127                     case 'y' : /* 'root' autolog at 2400 bauds */
128                                     autolog(tname);
129                     default: break;
130             }
131             for (;;) {
132                     for(tabp = itab; tabp < &itab[NITAB]; tabp++)
133                             if(tabp->tname == tname)
134                                     break;
135                     if(tabp >= &itab[NITAB])
136                             tabp = itab;
137                     tmode.sgispd = tabp->ispeed;
138                     tmode.sgospd = tabp->ospeed;
139                     tmode.sgflag = tabp->iflags;
140                     tmode.sgispd = tabp->ispeed;
141                     tmode.sgospd = tabp->ospeed;
142                     stty(0, &tmode);
143                     puts(tabp->message);
144                     stty(0, &tmode);
145                     if(getname()) {
146                             tmode.sgerase = ERASE;
147                             tmode.sgkill = KILL;
148                             tmode.sgflag = tabp->fflags;
149                             if(crmod)
150                                     tmode.sgflag =| CRMOD;
151                             if(upper)
152                                     tmode.sgflag =| LCASE;
153                             if(lower)
154                                     tmode.sgflag =& ~LCASE;
155                             stty(0, &tmode);
156                             execl('/bin/login', 'login', name, 0);
157                             exit(1);
158                     }
159                     tname = tabp->nname;
160             }
161     }
162
163     getname()
164     {
165             register char *np;
166             register c;
167             static cs;
168
169             crmod = 0;
170             upper = 0;
171             lower = 0;
172             np = name;
173             do {
174                     if (read(0, &cs, 1) <= 0)
175                             exit(0);
176                     if ((c = cs&0177) == 0)
177                             return(0);
178                     write(1, &cs, 1);
179                     if (c>='a' && c <='z')
180                             lower++;
181                     else if (c>='A' && c<='Z') {
182                             upper++;
183                             c =+ 'a'-'A';
184                     } else if (c==ERASE) {
185                             if (np > name)
186                                     np--;
187                             continue;
188                     } else if (c==KILL) {
189                             np = name;
190                             continue;
191                     }
192                     *np++ = c;
193             } while (c!='\n' && c!='\r' && np <= &name[16]);
194             *--np = 0;
195             if (c == '\r') {
196                     write(1, "\n", 1);
197                     crmod++;
198             } else
199                     write(1, "\r", 1);
200             return(1);
201     }
202
```

```
203     puts(as)
204     char *as;
205     {
206             register char *s;
207
208             s = as;
209             while (*s)
210                     write(1, s++, 1);
211     }
212     autolog(nn)
213     int nn;
214     {
215             switch(nn){
216                     case 'a':
217                     case 'd' :
218                     case 'x' :
219                             tmode.sgispd = B110;
220                             tmode.sgospd = B110;
221                             tmode.sgflag = ANYP+ECHO+XTABS+CR1+CRMOD;
222                             break;
223                     case 'b':
224                             tmode.sgispd = B300;
225                             tmode.sgospd = B300;
226                             tmode.sgflag = ANYP+ECHO+CR1+XTABS+CRMOD;
227                             break;
228                     case 'c':
229                     case 'e' :
230                     case 'y' :
231                             tmode.sgispd = B2400;
232                             tmode.sgospd = B2400;
233                             tmode.sgflag = ANYP+XTABS+ECHO+CRMOD+FF1;
234                             break;
235             }
236             tmode.sgerase = ERASE;
237             tmode.sgkill = KILL;
238             stty(0, &tmode);
239             switch(nn){
240                    case 'a' :
241                    case 'b' :
242                    case 'c' :
243                             write(1, "user logged in.\n", 16);
244                             execl("/bin/login", "login", "user", 0);
245                             break;
246                    case 'd' :
247                    case 'e' :
248                             write(1, "guest logged in.\n", 17);
249                             execl("/bin/login", "login", "guest", 0);
250                             break;
251                    case 'x' :
252                    case 'y' :
253                             write( 1, "root logged in.\n", 16);
254                             execl("/bin/login", "login", "root", 0);
255                             break;
256              }
257             printf("\n\nNO LOGGING POSSIBLE !!!!!!!!!!!!!!\n\n");
258             exit(1);
259     }
260
```

```
1       / disk boot program to load and transfer
2       / to a unix file system entry
3       / must be assembled with tty.s and
4       / appropriate disk driver
5
6       /       this is a special boot program:
7       /       it automatically loads the system
8       /       called TUNIX in the root
9       /       directory
10
11      / entry is made by jsr pc,*$0
12      / so return can be rts pc
13
14      core = 24.      / first core loc (in KW) not used
15      .. = [core*2048.]-512.
16      start:
17
18      / copy self to 'core' - 512. bytes
19      / strip of UNIX execute header if present
20
21              mov     $..,sp
22              mov     sp,r1
23              clr     r0
24              cmp     pc,r1
25              bhis    2f
26              cmp     (r0),$407
27              bne     1f
28              mov     $20,r0
29      1:
30              mov     (r0)+,(r1)+
31              cmp     r1,$core*2048.
32              blo     1b
33              jmp     (sp)
34
35      / clear all of core
36
37      2:
38              clr     (r0)+
39              cmp     r0,sp
40              blo     2b
41
42
43
44      /       no prompt
45      /       no path name read in
46      /       system tunix to be loaded
47
48              mov     $trvect,r5
49              mov     $names,r2
50              mov     r2,r1
51              movb    $'t,(r1)+
52              movb    $'u,(r1)+
53              movb    $'n,(r1)+
54              movb    $'i,(r1)+
55              movb    $'x,(r1)+
56      / start of path name decoding
57      / start with first name  and root ino
58
59              mov     $names,r2
60              mov     $1,r0
61
62      / get next inode
63
64      1:
65              clr     bno
66              jsr     pc,iget
67              tst     (r2)
68              beq     1f
69
70      / read next directory looking for next name
71
72      2:
73              jsr     pc,rmblk
74                      br start
75              mov     $buf,r1
76      3:
77              mov     r2,r3
78              mov     r1,r4
```

```
79              add     $16.,r1
80              tst     (r4)+
81              beq     5f
82      4:
83              cmpb    (r3)+,(r4)+
84              bne     5f
85              cmp     r4,r1
86              blo     4b
87              mov     -16.(r1),r0
88              add     $14.,r2
89              br      1b
90      5:
91              cmp     r1,$buf+512.
92              blo     3b
93              br      2b
94
95      / last entry was found
96      / read into 0.
97
98      1:
99              clr     r2
100     1:
101             jsr     pc,rmblk
102                     br callout
103             mov     $buf,r1
104     2:
105             mov     (r1)+,(r2)+
106             cmp     r1,$buf+512.
107             blo     2b
108             br      1b
109
110     / subroutine will read in inode
111     / number specified in r0
112     iget:
113             add     $31.,r0
114             mov     r0,r5
115             ash     $-4.,r0
116             jsr     pc,rblka
117             bic     $!17,r5
118             ash     $5.,r5
119             add     $buf,r5
120             mov     $inod,r4
121     1:
122             mov     (r5)+,(r4)+
123             cmp     r4,$addr+16.
124             blo     1b
125             rts     pc
126
127     / routine to read in block
128     / number specified by bno
129     / after applying file system
130     / mapping algorithm in inode.
131     / bno is incremented, success
132     / return is a skip, error (eof)
133     / is direct return.
134     rmblk:
135             add     $2,(sp)
136             mov     bno,r0
137             inc     bno
138             bit     $LRG,mode
139             bne     1f
140             asl     r0
141             mov     addr(r0),r0
142             bne     rblka
143     2:
144             sub     $2,(sp)
145             rts     pc
146     / large algorithm
147     / huge algorithm is not implemented
148     1:
149             clr     -(sp)
150             movb    r0,(sp)
151             clrb    r0
152             swab    r0
153             asl     r0
154             mov     addr(r0),r0
155             beq     2b
156             jsr     pc,rblka
157             mov     (sp)+,r0
158             asl     r0
159             mov     buf(r0),r0
160             beq     2b
161     rblka:
162             mov     r0,dska
163             br      rblk
```

```
164
165     ba:     buf
166     wc:     -256.
167     .bss
168     end:
169     inod = ..-1024.
170     mode = inod
171     addr = inod+8.
172     buf = inod+32.
173     bno = buf+514.
174     dska = bno+2
175     names = dska+2
176     LRG = 10000
177     .text
178
```

Annexe A.3

Procédures et Programmes de Contrôle

A.3.1 Procédure de contrôle "start"


```
1       echo AUTOMATIC BENCHMARK SYSTEM INITIALIZATION
2       echo              mkbench has to be runned before starting this!!!
3       : this shell procedure may only be executed by the root super-user
4
5       : set automatic logging on selected command tty
6       : set automatic logging on selected receiver ttys
7       ed /etc/ttys
8       /8/c
9       18a
10      .
11      /9/c
12      19e
13      .
14      /1/c
15      11e
16      .
17      /m/c
18      1me
19      .
20      /n/c
21      1ne
22      .
23      /f/c
24      1fy
25      .
26      /k/c
27      1ky
28      .
29      w
30      q
31      ed /etc/passwd
32      a
33      user::15:1::/apm:loop
34      .
35      w
36      q
37      cc -O alter.c
38      mv a.out alter
39
```

A.3.2 Procédure de contrôle "loop"

```
1       :
2       :
3       :       AUTOMATIC PERFORMANCE MEASUREMENT SYSTEM
4       :
5       :                       MAIN CONTROL LOOP
6       :
7       :
8       echo    ++++++++ loop entered
9       date
10      echo    ++++++++ initiating asynchronous flows
11      sh bench/t4/flow4 &
12      sh bench/t3/flow3 &
13      sh bench/t2/flow2 &
14      echo    ++++++++ initiating measured flow
15      date
16      sh bench/t1/flow1
17      skill
18      date
19      echo    ++++++++ measured flow terminated
20      echo    ++++++++ regenerating benchmark files
21      time sh bench/majfb14
22      echo    ++++++++ all benchmark files regenerated
23      date
24      rm -f /usr/lpd/*
25      rm -f /tmp/*
26      echo    ++++++++ preparing the new system
27      updval
28      alter
29      cp tune.h /usr/sys/tune.h
30      echo    ++++++++ new parameter values are :
```

```
31      cat tune.h
32      time sh newsys
33      echo    ++++++++ new system ready
34      date
35      echo    ++++++++ checking the disks :
36      icheck -s /dev/hp0?
37      echo    ++++++++ booting again
38      reboot
39
```

A.3.3 Procédure de contrôle "stop"

```
1       :
2       : restore normal use of terminals.
3       :
4       ed /etc/ttys
5       /8/c
6       18-
7       .
8       /9/c
9       1g2
10      .
11      /l/c
12      112
13      .
14      /m/c
15      1m2
16      .
17      /n/c
18      1n2
19      .
20      /f/c
21      1f2
22      .
23      /k/c
24      1k2
25      .
26      w
27      q
28      :       remove special user.
29      ed /etc/passwd
30      /user/d
31      w
32      q
33      :       restore original operating system
34      :               and reboot it.
35      cp /unix450 /tunix
36      reboot
```

A.3.4 Procédure de contrôle "newsys"

```
1       :
2       :
3       :       compile, archive and load the new operating system
4       :
5       :
6       :
7       chdir /usr/sys/ken
8       cc -c -O *.c
9       ar r ../lib1 *.o
10      rm *.o
11      chdir ../dmr
12      cc -c -O *.c
13      ar r ../lib2 *.o
14      rm *.o
15      chdir ../conf
16      ld -x -r -d l45.o m45.o c450.o ../lib1 ../lib2
17      nm -ug
18      sysfix a.out /tunix
```

A.3.5 Procédure de contrôle majfb14

```
1       :
2       :
3       :               restore  all  benchmark files
4       :
5       :
6       :
7       chdir bench
8       cp 2afpi25.f t2/afpi25.f
9       cp 3afpi25.f t3/afpi25.f
10      cp 4afpi25.f t4/afpi25.f
11      cp afpi1.f t2/afpi1.f
12      cp afpi1.f t3/afpi1.f
13      cp afpi1.f t4/afpi1.f
14      cp 2afpi3.f t2/afpi3.f
15      cp 3afpi3.f t3/afpi3.f
16      cp 4afpi3.f t4/afpi3.f
17      cp afpi5.c t2/afpi5.c
18      cp afpi5.c t3/afpi5.c
19      cp afpi5.c t4/afpi5.c
20      cp 2afpi6.c t2/afpi6.c
21      cp 3afpi6.c t3/afpi6.c
22      cp 4afpi6.c t4/afpi6.c
23      cp entree71 t2/entree7
24      cp entree71 t3/entree7
25      cp entree71 t4/entree7
26      cp 1afpi25.f t1/afpi25.f
27      cp afpi1.f t1/afpi1.f
28      cp 1afpi3.f t1/afpi3.f
29      cp afpi5.c t1/afpi5.c
30      cp 1afpi6.c t1/afpi6.c
31      cp entree71 t1/entree7
32      chdir /apm
```

A.3.6 Programme de planification "mkbench.c"

```
1       #
2
3
4       /*      FUN79-APM
5
6               mkbench program
7
8
9               mkbench asks the user for the datas of the experimental design
10              he wishes to run.
11              then, it computes the design by crossing completely every level
12              of a factor with each level of the others.
13              the output of this program are the files 'values' and 'params.c'
14
15              'values.c' represents the experimental design. each lines contains
16              the parameter values for one experience. if replications
17              are desired, they are incorporated automatically.
18
19              'params.c' is a part of a C-program, containing only the
20              declaration of the parameter name array. this file is for use
21              of the program alter.c, which has to be recompiled after production
22              of 'params.c'
23
24              Nota: input is interactive.
25
26
27      */
28      #include 'lib.c'
29
30
31      #define MAXFACTS        5
32      #define MAXLEVELS       3
33      #define MAXREPLICS      3
34
35      #define newline printf('\n')
36      #define comma   printf(',');
```

```
#define quote   printf("\"")
#define tab     printf("\t")

#define eachfactor      v = 0; v < FACTS; v++
#define eachlevelB      m = 0; m < levels[0]; m++
#define eachlevelC      l = 0; l < levels[1]; l++
#define eachlevelD      k = 0; k < levels[2]; k++
#define eachlevelE      J = 0; J < levels[3]; J++
#define eachlevelF      i = 0; i < levels[4]; i++
#define eachreplic      n = 0; n < REPLICS; n++
#define eachlevel       fl = 0; fl < levels[v]; fl++


char    *str1   "#define        NFACTS  ";
char    *str2   "char *names[] { ";
char    *str3   "};";

char    *file1  "params.c";
char    *file2  "values";
int     fd1;
int     fd2;
int     fdstd;
int     FACTS;
int     REPLICS;
int     levels[MAXFACTS];
char    name[MAXFACTS][15];
int     value[MAXFACTS][MAXLEVELS];


main()
{
        register int v, m, l;
        int i, J, k, n, fl;
        int EXP;


        printf("CROSSED DATA DESIGN: \n\nHow much factors ?\t\t\t");
        /*
         * first read in the number of factors.
         */
        if(((FACTS = readint(0)) <= 0) || (FACTS > MAXFACTS)){
                printf("ERROR : bad number of factors\n\n");
                exit();
        }
        /*
         * read in name and level values for each factor
         */
        for(eachfactor){
                printf("Factor  %c :\tName : \t\t\t", v-0+'A');
                readstr(0, name[v]);
                printf("\t\tHow much levels ?\t");
                if(((levels[v] = readint(0)) <= 0) || (levels[v] > MAXLEVELS)) {
                        printf("ERROR : bad number of levels\n\n");
                        exit();
                }
                for(eachlevel){
                        printf("\t\t\tLevel %d :\tValue ?\t", fl);
                        value[v][f]] = readint(0);
                }
        }
        /*
         * now, read in number of replications
         */
        printf("\nHow much replications in this design ?\t");
        if(((REPLICS = readint(0)) < 0) || (REPLICS > MAXREPLICS)){
                printf("ERROR : bad number of replications\n\n");
                exit();
        }

        /*
         *  create all files
         */
        close(creat(file1, 0666));
        close(creat(file2, 0666));


        /*
         *  write the files
         */
        fdstd = dup(1);
        close(1);
        fd1 = open(file1, 1);
        printf("%s%d\n", str1, FACTS);
```

```
123             printf("%s\n", str2);
124             for(eachfactor){
125                     tab;
126                     quote;
127                     printf("%s", name[v]);
128                     quote;
129                     if( v != FACTS-1) comma;
130                     newline;
131             }
132             printf("%s\n", str3);
133             close(fd1);
134
135             fd2 = open(file2, 1);
136             switch(FACTS) {
137
138                     case 1 :        for(eachlevelB)
139                                     for(eachreplic)
140                                     printf("%d\n", value[0][m]);
141                                     break;
142                     case 2 :        for(eachlevelB)
143                                     for(eachlevelC)
144                                     for(eachreplic)
145                                     printf("%d  %d\n", value[0][m], value[1][l]);
146                                     break;
147
148                     case 3 :        for(eachlevelB)
149                                     for(eachlevelC)
150                                     for(eachlevelD)
151                                     for(eachreplic)
152                                     printf("%d  %d  %d\n", value[0][m], value[1][l],
153                                             value[2][k]);
154                                     break;
155
156                     case 4 :        for(eachlevelB)
157                                     for(eachlevelC)
158                                     for(eachlevelD)
159                                     for(eachlevelE)
160                                     for(eachreplic)
161                                     printf("%d  %d  %d  %d\n", value[0][m],
162                                             value[1][l], value[2][k], value[3][j]);
163                                     break;
164
165                     case 5 :        for(eachlevelB)
166                                     for(eachlevelC)
167                                     for(eachlevelD)
168                                     for(eachlevelE)
169                                     for(eachlevelF)
170                                     for(eachreplic)
171                                     printf("%d  %d  %d  %d  %d\n", value[0][m],
172                                             value[1][l], value[2][k], value[3][j],
173                                             value[4][i]);
174                                     break;
175
176             }
177
178             close(fd2);
179             dup(fdstd);
180             close(fdstd);
181
182             /*
183              *  compute number of experiences of the design.
184              *  print it out for information.
185              */
186             switch(FACTS) {
187
188                     case 1 :        EXP = levels[0]*REPLICS;
189                                     break;
190
191                     case 2 :        EXP = levels[1]*levels[0]*REPLICS;
192                                     break;
193
194                     case 3 :        EXP = levels[2]*levels[1]*levels[0]*REPLICS;
195                                     break;
196
197                     case 4 :        EXP = levels[3]*levels[2]*levels[1]*levels[0]
198                                             *REPLICS;
199                                     break;
200
201                     case 5 :        EXP = levels[4]*levels[3]*levels[2]*levels[1]
202                                             *levels[0]*REPLICS;
203                                     break;
204             }
205
206             printf("\n\n%d experiences are necessary for this design\n", EXP);
207             printf("mkbench done\n");
208     }
```

```
1
2
3
4       /*
5
6               FUN79-APM
7
8               updval program
9
10
11              this program performs the updating of the parameter values file
12              'values', created by mkbench.
13
14              it reads the first line of the file. if it is also the last one,
15              it removes the file and executes the shell procedure 'stop'.
16
17              if it is not the last line, it restores the truncated file
18              and exits normally.
19
20      */
21
22      char *temp '/tmp/updv';
23      char *file 'values';
24
25
26      main()
27      {
28
29              register  int tmpf, fd;
30              register char *bp;
31              char buffer[100];
32              int count, linec, rd;
33
34
35              bp = buffer;
36              linec = 0;
37              count = 0;
38
39              /*
40               *      create temporary file.
41               */
42              if((tmpf = creat(temp, 0666)) == -1) {
43                      printf('Can't create temp file\n');
44                      goto stop;
45              }
46              /*
47               *      open 'values' file.
48               */
49              if((fd = open(file, 0)) == -1) {
50                      printf('Can't find values file\n');
51                      goto stop;
52              }
53
54      inline :
55
56              /*
57               *      read in.
58               */
59              while(((rd = read(fd, bp, 1)) != 0) && ( rd != -1) && (count < 100)){
60                      count++;
61                           /*
62                            *      if new line was found, process the old one.
63                            */
64                      if(*bp == '\n') { linec++; goto endline; }
65                      bp++;
66              }
67              goto endfile;
68
69      endline :
70
71              bp++;
72              /*
73               *      if we passed the first line, copy all the following ones
74               *    . into the temporary file.
75               */
76              if(linec > 1) { write(tmpf, buffer, count);}
77              count = 0;
78              bp = buffer;
79              goto inline;
```

```
80
81      endfile :
82
83              if(linec > 1) {
84                      close(fd); close(tmpf);
85                          /*
86                           *      restore the truncated 'values' file .
87                           */
88                      execl("/bin/mv", "mv", temp, file, 0);
89                      printf("Can't move temp file\n");
90                      goto out;
91              }
92
93      stop :
94
95
96              close(fd); close(tmpf);
97              /*
98               *      remove 'values' and temporary files.
99               */
100             unlink(temp); unlink(file);
101             /*
102              *      try until the end of the world to execute stop shell.
103              */
104             for(;;) execl("/bin/sh", "sh", "stop", 0);
105
106     out :  ;
107
108     }
```

A.3.8 Programme de mise à jour du fichier "tune.h", "alter.c"

```
1       #
2
3
4
5       /*
6               FUN79-APM
7
8               alter program
9
10
11              this program performs the updating of the system parameter
12              declarations file 'tune.h'.
13
14              it reads the files 'values.c', the first line is supposed to
15              contain the right values for the next experience.
16
17              then, it rewrites the file 'tune.h' with the new parameter values.
18
19
20              this program includes, at his compilation, the file 'params.c',
21              which has to contain the declarations of 'NFACTS', which is the
22              number of parameters, and of the array 'names', with the parameter
23              names. this file is produced by the mkbench program; so 'alter.c'
24              has to be compiled after running mkbench.
25
26
27              since alter is not likely to live in the same directory as 'tune.h',
28              that is, '/usr/sys', it should work on a copy of this file. this
29              copy can then be moved by means of 'cp' or 'mv' commands.
30
31
32      */
33      #include        "params"
34
35      char *str       "#define";
36      char buffer[40];
37      char ascii[10];
38
39      main()
40      {
41
42              extern char *names[], ascii[], buffer[];
43              register char **np, *bp, *vp;
44              char *stp;
45              int fd1, fd2, i;
46
47              np = names;
48              vp = ascii;
```

```
49              fd1 = open('values', 0);
50              fd2 = open('tune', 1);
51
52              for( i = 1; i <= NFACTS; i++){
53                      clear(buffer, 40);
54                      clear(ascii, 10);
55                      getval(fd1, vp);
56                      bp = buffer;
57                      str = str;
58                      strcopy(str, bp, 7);
59                      bp =+ 10;
60                      *bp++ = '\t';
61                      strcopy(*np, bp, 7);
62                      bp =+ 10;
63                      *bp++ = '\t';
64                      strcopy(vp, bp, 10);
65                      buffer[39] = '\n';
66                      write(fd2, buffer, 40);
67                      np++;
68                      vp = ascii;
69                      }
70      }
71      strcopy(s1, s2, l)
72      char *s1, *s2;
73      int l;
74      {
75              int i;
76              for(i = 0; i<l; i++) *s2++ = *s1++;
77      }
78      getval(fd, loc)
79      int fd;
80      char *loc;
81      {
82              char *c;
83              *c = ' ';
84              while( (*c == ' ') || (*c == '\t')){ read(fd, c, 1);}
85              while( (*c != ' ') && (*c != '\t')){
86                      if( *c == '\n') return(0);
87                      *loc++ = *c;
88                      read(fd, c, 1);
89                      }
90              return(1);
91      }
92      clear(s, l)
93      char s[];
94      int l;
95      {
96              int i;
97              for(i = 0; i<l; i++) s[i] = ' ';
98      }
99
```

A.3.9 Programme de collecte des données "tdc.c"

```
1
2
3
4       /*      FUN79-APM
5
6
7               tdc program
8
9
10
11              this is the program which reads a file like bt1, produced by
12              the benchmark control procedure.
13
14              the file may contain records like those produced by the 'time'
15              command prefix, intermixed with lines of garbage.
16              tdc selectively reads only the lines starting with 'real',
17              to catch the real execution times of the
18              benchmark programs.
19
20              the program converts the reported time string into a floating
21              number of seconds.
22              these numbers are summarized and the mean is
23              computed.
```

```
24
25              this mean is written at the end of the file 'means'.
26              tdc prints out the total execution time (which is not the elapsed
27              time of the run of the whole benchmark procedure!), the number
28              of time records found in the file and the computed mean
29              execution time.
30
31              the records in the file 'means' are written one per line, in %.6f
32              format.
33
34
35      */
36      int fd;
37      float tt;
38      float gt;
39      int count;
40
41
42      main()
43      {
44
45              int rd, inr;
46              char filename[20];
47
48              printf('Time Data Collection.\n\n');
49              printf('on which file are the datas ?\t');
50              rd = read( 0, filename, 20);
51              filename[rd-1] = '\0';
52
53              inr = dup(0);
54              close(0);
55              if((fd = open(filename, 0)) == -1) {
56                      printf('ERROR: file not found.\n');
57                      exit();
58              }
59
60              input();
61              close(fd);
62              dup(inr);
63              close(inr);
64              file();
65              printf('total time is \t %.2f\n',  tt);
66              printf('count is \t %d\n', count);
67              printf('mean  time is \t %.2f\n', gt);
68              printf('tdc done\n');
69      }
70
71
72      input()
73      {
74
75              int lp, outr;
76              float atot();
77              float time;
78              char timbuf[7];
79              char c;
80
81              count = 0;
82              tt = 0.0;
83              outr = dup(1);
84              close(1);
85              lp = open('/dev/lp', 1);
86
87              for(;;){
88
89                      if(read(fd, &c, 1) == 0) break;
90                      if(c == 'r'){
91                        read(fd, &c, 1);
92                        if(c == 'e'){
93                          read(fd, &c, 1);
94                          if(c == 'a'){
95                            read(fd, &c, 1);
96                            if(c == 'l'){
97                              seek(fd, 4, 1);
98                              if(read(fd, timbuf, 7) == 0){
99                                      write(2, 'ERROR: unexpected EOF\n', 21);
100                                     exit();
101                             }
102                             time = atot(timbuf);
103                             tt =+ time;
104                             count =+ 1;
105                             printf('%d\t%f\n', count, time);
106                             seek(fd, 25, 1);
```

```
107                         }
108                       }
109                     }
110                   }
111             }
112             close(lr);
113             dup(outr);
114             close(outr);
115             if( count == 0){
116                     printf("ERROR : no times on file\n");
117                     exit();
118             }
119             if(tt <= 0.005){
120                     printf("ERROR : zero total time\n");
121                     exit();
122             }
123             st = tt / count;
124     }
125
126     file()
127     {
128             int outr;
129             int fdm;
130             int statb[18];
131             outr = dup(1);
132             close(1);
133             if(stat("means", &statb) < 0){
134                     close(creat("means", 0666));
135             }
136             fdm = open("means", 1);
137             seek(fdm, 0, 2);
138             printf("%f\n", st);
139             close(fdm);
140             dup(outr);
141     }
142
143
144     atof(s)
145     char s[];
146     {
147             char v[2];
148             int min, sec, ten;
149             float t;
150             v[0] = s[0]; v[1] = s[1];
151             min = atoi(v);
152             v[0] = s[3]; v[1] = s[4];
153             sec = atoi(v);
154             ten = atoi(s[6]);
155
156             t = min * 60.0 + sec * 1.0 + ten/10.0;
157             return(t);
158     }
159
```

```
/*      FUN79-APM


        anova programs



        anova is a set of programs for performing an analysis of
        variance for a completely crossed parametric design of one
        through five factors.

        essentially, the analysis of variance is the computing of
        sums of squares and degrees of freedom.

        the second step of  the analysis of variance id the interpretation
        of those sums of squares, which can be done by the F-test.
        this second part could have been implemented, but we thaught it
        was'nt worth it.


        this analysis of variance is implemented by a set of programs.
        it allows a modular conception of programming, and the addition
        of some parts, such as one more factor, can be done quite easily.

        the first part is the main decision program "anova.c"
        which asks for the number of factors and calls the corresponding
        analysis module. there are five analysis modules, one for each
        allowed design.  these modules first read in the datas  (procedure
        input). other input is interactive. then they compute the sums of
        squares (we wouldn't describe here how this works. it's quite easy to
        understand.) and the degrees of freedom. once this is done, they
        write a temporary file and call the printout module.
        this last module reads the temporary file and lists  the table
        of observations ant the analysis of variance table.



*/
```

a - procédure de compilation

```
cc -O anova.c
mv a.out anova
cc -f -O test1.c
mv a.out test1
cc -f -O test2.c
mv a.out test2
cc -f -O test3.c
mv a.out test3
cc -f -O test4.c
mv a.out test4
cc -f -O test5.c
mv a.out test5
cc -f -O printout.c
mv a.out printout
echo run done
```

```
1       #
2
3
4
5       /*   sum  of  squares
6       ************************
7
8
9          needed for                    1-way   2-way   3-way   4-way   5-way classific.
10                                       :       :       :       :       :
11                                       v       v       v       v       v
12
13      */
14      /*-------------------------------------------------------------------------------*/
15      float   TSS      0.0;    /*      :       :       :       :       :       */
16      float   SS_ADD   0.0;    /*      :       :       :       :       :       */
17      float   SS_CELL  0.0;    /*      :       :       :       :       :       */
18      /*-------------------------------------------------------------------------------*/
19      float   SS_B     0.0;    /*      :       :       :       :       :       */
20      /*-------------------------------------------------------------------------------*/
21      float   SS_C     0.0;    /*              :       :       :       :       */
22      float   SS_BC    0.0;    /*              :       :       :       :       */
23      /*-------------------------------------------------------------------------------*/
24      float   SS_D     0.0;    /*                      :       :       :       */
25      float   SS_BD    0.0;    /*                      :       :       :       */
26      float   SS_CD    0.0;    /*                      :       :       :       */
27      float   SS_BCD   0.0;    /*                      :       :       :       */
28      /*-------------------------------------------------------------------------------*/
29      float   SS_E     0.0;    /*                              :       :       */
30      float   SS_BE    0.0;    /*                              :       :       */
31      float   SS_CE    0.0;    /*                              :       :       */
32      float   SS_DE    0.0;    /*                              :       :       */
33      float   SS_BCE   0.0;    /*                              :       :       */
34      float   SS_BDE   0.0;    /*                              :       :       */
35      float   SS_CDE   0.0;    /*                              :       :       */
36      float   SS_BCDE  0.0;    /*                              :       :       */
37      /*-------------------------------------------------------------------------------*/
38      float   SS_F     0.0;    /*                                      :       */
39      float   SS_BF    0.0;    /*                                      :       */
40      float   SS_CF    0.0;    /*                                      :       */
41      float   SS_DF    0.0;    /*                                      :       */
42      float   SS_EF    0.0;    /*                                      :       */
43      float   SS_BCF   0.0;    /*                                      :       */
44      float   SS_BDF   0.0;    /*                                      :       */
45      float   SS_BEF   0.0;    /*                                      :       */
46      float   SS_CDF   0.0;    /*                                      :       */
47      float   SS_CEF   0.0;    /*                                      :       */
48      float   SS_DEF   0.0;    /*                                      :       */
49      float   SS_BCDF  0.0;    /*                                      :       */
50      float   SS_BCEF  0.0;    /*                                      :       */
51      float   SS_BDEF  0.0;    /*                                      :       */
52      float   SS_CDEF  0.0;    /*                                      :       */
53      float   S_BCDEF  0.0;    /*                                      :       */
54      /*-------------------------------------------------------------------------------*/
55
56
57
58
59      #define MAXFACTS          5       /*maximum number of factors.
60                                          MAY NOT BE CHANGED ! */
61      #define MAXLEVELS         3       /*maximum number of levels*/
62      #define MAXREPLICS        3       /*maximum number of replications*/
63
64      int     levels[5]         {               /*actual level numbers:         */
65                                0,              /*index 0 stands for factor B */
66                                0,              /*index 1 stands for factor C */
67                                0,              /*index 2 stands for factor D */
68                                0,              /*index 3 stands for factor E */
69                                0               /*index 4 stands for factor F */
70                              };
71      int     REPLICS           0;
72      int     FACTS             0;
73
74      float x[MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXREPLICS];
75
```

```
76
77
78      #define eachfactor      v  = 0; v < FACTS; v++
79      #define eachlevelB      m  = 0; m < levels[0]; m++
80      #define eachlevelC      l  = 0; l < levels[1]; l++
81      #define eachlevelD      k  = 0; k < levels[2]; k++
82      #define eachlevelE      j  = 0; j < levels[3]; j++
83      #define eachlevelF      i  = 0; i < levels[4]; i++
84      #define eachreplic      n  = 0; n < REPLICS; n++
85      #define x1              x[0][0][0][0]
86      #define x2              x[0][0][0]
87      #define x3              x[0][0]
88      #define x4              x[0]
89
90      char    factor[MAXFACTS][20];
91      char filename[20];                      /* name of input file */
```

c - module principal "main.c"

```
1       #
2
3       #include "lib.c"
4       main()
5       {
6               register int f;
7               printf("How many factors? \t");
8               f = readint(0);
9               switch(f){
10              case 0: printf("You like jokes I see!\n"); break;
11              case 1: execl("test1", 0); printf("Can't find test1.Breaking\n");
12                      break;
13              case 2: execl("test2", 0); printf("Can't find test2.Breaking\n");
14                      break;
15              case 3: execl("test3", 0); printf("Can't find test3.Breaking\n");
16                      break;
17              case 4: execl("test4", 0); printf("Can't find test4.Breaking\n");
18                      break;
19              case 5: execl("test5", 0); printf("Can't find test5.Breaking\n");
20                      break;
21              default: printf("%d-way analysis is'nt implemented.\n", f);
22              }
23      }
24
```

d - module d'analyse, plan à un facteur, "test1.c"

```
1       #include "lib.c"
2       #include "spec.h"
3       #include "input.c"
4       #include "out.c"
5
6       main()
7       {
8               FACTS = 1;
9               input();
10              oneway();
11              out();
12              execl("printout", 0);
13              printf("Can't find printout!\n");
14
15      }
16
17
```

```
18      oneway()
19      {
20
21              register  int m, n;             /* indexes */
22              float oat;                      /* over all total */
23              float ct;                       /* cell total */
24              float temp;
25
26              oat = 0.0;
27
28              for(eachlevelB) {
29                      ct = 0.0;
30                      for(eachreplic) {
31                              temp = x1[m][n];
32                              ct =+ temp;
33                              TSS =+ square(temp);
34                      }
35                      SS_B =+ square(ct);
36                      oat =+ ct;
37              }
38
39              SS_CELL = square(oat) / (levels[0] * REPLICS);
40              SS_B =/ REPLICS;
41      }
```

e - module d'analyse, plan à 2 facteurs, "test 2.c"

```
1       #
2       #include "lib.c"
3       #include "spec.h"
4       #include "input.c"
5       #include "out.c"
6
7       main()
8       {
9               FACTS = 2;
10              input();
11              twoway();
12              out();
13              execl("printout", 0);
14              printf("Can't find printout! \n");
15      }
16
17
18
19
20      twoway()
21      {
22
23              register int m, l, n;
24              float oat;                      /* over-all total */
25              float ct;                       /* cell total */
26              float febt;                     /* for each b total */
27              float fect[MAXLEVELS];          /* for each c totals */
28              float temp;                     /* temporary */
29
30              for(eachlevelC) {
31                      fect[l] = 0.0 ;
32              }
33              oat = 0.0;
34
35              for(eachlevelB) {
36                      febt = 0.0;
37                      for(eachlevelC) {
38                              ct = 0.0;
39                              for(eachreplic) {
40                                      temp = x2[l][m][n];
41                                      TSS =+ square(temp);
42                                      ct =+ temp;
43                              }
44                              febt =+ ct;
45                              fect[l] =+ ct;
46                      SS_BC =+ square(ct);
47                      }
48                      SS_B =+ square(febt);
49                      oat =+ febt;
50              }
51
52              for(eachlevelC) {
53                      SS_C =+ square(fect[l]);
54              }
55
56              SS_CELL = square(oat) / (levels[0] * levels[1] * REPLICS );
```

```
57              SS_B  =/  (levels[1] * REPLICS );
58              SS_C  =/  (levels[0] * REPLICS );
59              SS_BC =/ REPLICS;
60
61      }
```

f - module d'analyse, plan à 3 facteurs, "test 3.c"

```
1       {
2
3       #include "lib.c"
4       #include "spec.h"
5       #include "input.c"
6       #include "out.c"
7
8
9       main()
10      {
11
12              FACTS = 3;
13              input();
14              threeway();
15              out();
16              execl("printout", 0);
17              printf("Can't find printout !\n");
18      }
19
20
21
22
23      #define CELL_TOTAL      y[k][l][m]
24
25
26      threeway()
27      {
28
29
30              register int k, l, m;
31              int n;
32
33              float temp, ct, oat, t1, t2;
34              float y[MAXLEVELS][MAXLEVELS][MAXLEVELS];
35
36
37              for(eachlevelB){
38                      t1 = 0.0;
39                      for(eachlevelC){
40                              t2 = 0.0;
41                              for(eachlevelD){
42                                      ct = 0.0;
43                                      for(eachreplic){
44                                              temp = x3[k][l][m][n];
45                                              ct =+ temp;
46                                              TSS =+ square(temp);
47                                      }
48                                      oat =+ ct;
49                                      SS_BCD =+ square(ct);
50                                      t2 =+ ct;
51                                      t1 =+ ct;
52                                      CELL_TOTAL = ct;
53                              }
54                              SS_BC =+ square(t2);
55                      }
56                      for(eachlevelD){
57                              t2 = 0.0;
58                              for(eachlevelC) t2 =+ CELL_TOTAL;
59                              SS_BD =+ square(t2);
60                      }
61                      SS_B =+ square(t1);
62              }
63
64              for(eachlevelC){
65                      t1 = 0.0;
66                      for(eachlevelD){
67                              t2 = 0.0;
68                              for(eachlevelB) { temp = CELL_TOTAL;
69                                              t2 =+ temp;
70                                              t1 =+ temp;
71                                              }
72                              SS_CD =+ square(t2);
73                      }
```

```
74                      SS_C =+ square(t1);
75              }
76
77              for(eachlevelD){
78                      t1 = 0.0;
79                      for(eachlevelC) for(eachlevelB) t1 =+ CELL_TOTAL;
80                      SS_D =+ square(t1);
81              }
82              SS_CELL = square(oat) / (levels[0]*levels[1]*levels[2]*REPLICS);
83              SS_B  =/  (levels[1]*levels[2]*REPLICS);
84              SS_C  =/  (levels[0]*levels[2]*REPLICS);
85              SS_D  =/  (levels[0]*levels[1]*REPLICS);
86              SS_BC =/  (levels[2]*REPLICS);
87              SS_CD =/  (levels[0]*REPLICS);
88              SS_BD =/  (levels[1]*REPLICS);
89              SS_BCD =/ REPLICS;
90
91      }
```

g - module d'analyse, plan à 4 facteurs, "test 4.c"

```
1       #
2
3       #include "lib.c"
4       #include "spec.h"
5       #include "input.c"
6       #include "out.c"
7
8
9       main()
10      {
11
12              FACTS = 4;
13              input();
14              fourway();
15              out();
16              execl("printout", 0);
17              printf("Can't find printout ! \n");
18      }
19
20
21      #
22
23      #define CELL_TOTAL      y[j][k][l][m]
24
25      fourway()
26      {
27
28
29              int j, k, l, m, n;
30              float temp, t1, t2, t3;
31              float y[MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXLEVELS];  /* cell totals */
32              float ct, oat;
33
34
35              for(eachlevelB){
36                      t1 = 0.0;
37                      for(eachlevelC){
38                              t2 = 0.0;
39                              for(eachlevelD){
40                                      t3 = 0.0;
41                                      for(eachlevelE){
42                                              ct = 0.0;
43                                              for(eachreplic){
44                                                      temp = x4[j][k][l][m][n];
45                                                      ct =+ temp;
46                                                      TSS =+ square(temp);
47                                              }
48                                              SS_BCDE =+ square(ct);
49                                              oat =+ ct;
50                                              y[j][k][l][m] =+ ct;
51                                              t3 =+ ct;
52                                              t2 =+ ct;
53                                              t1 =+ ct;
54                                      }
55                                      SS_BCD =+ square(t3);
56                              }
```

```
                        SS_BC =+ square(t2);
                        for(eachlevelE){
                                t3 = 0.0;
                                for(eachlevelD) t3 =+ CELL_TOTAL;
                                SS_BCE =+ square(t3);
                        }
                }
                SS_B =+ square(t1);
                for(eachlevelD){
                        t2 = 0.0;
                        for(eachlevelE){
                                t3 = 0.0;
                                for(eachlevelC){
                                        temp = CELL_TOTAL;
                                        t3 =+ temp;
                                        t2 =+ temp;
                                }
                                SS_BDE =+ square(t3);
                        }
                        SS_BD =+ square(t2);
                }
                for(eachlevelE){
                        t2 = 0.0;
                        for(eachlevelD) for(eachlevelC) t2 =+ CELL_TOTAL;
                        SS_BE =+ square(t2);
                }
        }

        for(eachlevelC){
                t1 = 0.0;
                for(eachlevelD){
                        t2 = 0.0;
                        for(eachlevelE){
                                t3 = 0.0;
                                for(eachlevelB){
                                        temp = CELL_TOTAL;
                                        t3 =+ ct;
                                        t2 =+ ct;
                                        t1 =+ ct;
                                }
                                SS_CDE =+ square(t3);
                        }
                        SS_CD =+ square(t2);
                }
                SS_C =+ square(t1);
                for(eachlevelE){
                        t2 = 0.0;
                        for(eachlevelD) for(eachlevelB) t2 =+ CELL_TOTAL;
                        SS_CE =+ square(t2);
                }
        }

        for(eachlevelD){
                t1 = 0.0;
                for(eachlevelE){
                        t2 = 0.0;
                        for(eachlevelC)
                                for(eachlevelB){
                                        temp = CELL_TOTAL;
                                        t2 =+ temp;
                                        t3 =+ temp;
                                }
                        SS_DE =+ square(t2);
                }
                SS_D =+ square(t1);
        }

        for(eachlevelE){
                t1 = 0.0;
                for(eachlevelD)
                  for(eachlevelC)
                    for(eachlevelB) t1 =+ CELL_TOTAL;
                SS_E =+ square(t1);
        }



        SS_CELL = square(oat) /(levels[0]*levels[1]*levels[2]*levels[3]*REPLICS);
        SS_B    =/ (levels[1]*levels[2]*levels[3]*REPLICS);
        SS_C    =/ (levels[0]*levels[2]*levels[3]*REPLICS);
        SS_D    =/ (levels[0]*levels[1]*levels[3]*REPLICS);
        SS_E    =/ (levels[0]*levels[1]*levels[2]*REPLICS);
```

```
141             SS_BC   =/ (levels[2]*levels[3]*REPLICS);
142             SS_BD   =/ (levels[1]*levels[3]*REPLICS);
143             SS_BE   =/ (levels[1]*levels[2]*REPLICS);
144             SS_CD   =/ (levels[0]*levels[3]*REPLICS);
145             SS_CE   =/ (levels[0]*levels[2]*REPLICS);
146             SS_DE   =/ (levels[0]*levels[1]*REPLICS);
147             SS_BCD  =/ (levels[3]*REPLICS);
148             SS_BCE  =/ (levels[2]*REPLICS);
149             SS_BDE  =/ (levels[1]*REPLICS);
150             SS_CDE  =/ (levels[0]*REPLICS);
151             SS_BCDE =/ REPLICS;
152
153     }
```

h - module d'analyse, plan à 5 facteurs, "test5.c"

```
1       #
2
3       #include "lib.c"
4       #include "spec.h"
5       #include "input.c"
6       #include "out.c"
7
8
9       main()
10      {
11
12              FACTS = 5;
13              input();
14              fiveway();
15              out();
16              execl("printout", 0);
17              printf("Can't find printout ! \n");
18      }
19
20
21      #
22
23
24      #define CELL_TOTAL      y[i][j][k][l][m]
25
26
27      fiveway()
28      {
29
30              float ct, oat, t1, t2, t3, t4, temp;
31              float y[MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXLEVELS][MAXLEVELS];
32                                              /*cell totals*/
33
34              register int k, l, m;
35              int i, j, n;
36
37
38
39
40
41              /* compute sum of squares. when the deepest 'for'
42                      is reached for  the first time, compute cell totals */
43
44
45
46
47      for(eachlevelB){
48              t1 = 0.0;
49              for(eachlevelC){
50                      t2 = 0.0;
51                      for(eachlevelD){
52                              t3 = 0.0;
53                              for(eachlevelE){
54                                      t4 = 0.0;
55                                      for(eachlevelF){
56                                              ct = 0.0;
57                                              for(eachreplic){
58                                                      temp = x[i][j][k][l][m][n];
59                                                      ct =+ temp;
60                                                      TSS =+ square(temp);
61                                              }
```

```
62                                              SS_BCDEF =+ square(ct);
63                                              cnt =+ ct;
64                                              CELL_TOTAL = ct;
65                                              t4 =+ ct;
66                                              t3 =+ ct;
67                                              t2 =+ ct;
68                                              t1 =+ ct;
69                                      }
70                                      SS_BCDE =+ square(t4);
71                              }
72                              SS_BCD =+ square(t3);
73                              for(eachlevelF){
74                                      t4 = 0.0;
75                                      for(eachlevelE) t4 =+ CELL_TOTAL;
76                                      SS_BCDF =+ square(t4);
77                              }
78                      }
79                      for(eachlevelE){
80                              t3 = 0.0;
81                              for(eachlevelF){
82                                      t4 = 0.0;
83                                      for(eachlevelD){
84                                              temp = CELL_TOTAL;
85                                              t4 =+ ct;
86                                              t3 =+ ct;
87                                      }
88                                      SS_BCEF =+ square(t4);
89                              }
90                              SS_BCE =+ square(t3);
91                      }
92                      for(eachlevelF){
93                              t3 = 0.0;
94                              for(eachlevelE) for(eachlevelD) t3 =+ CELL_TOTAL;
95                              SS_BCF =+ square(t3);
96                      }
97                      SS_BC =+ square(t2);
98              }
99              for(eachlevelD){
100                     t2 = 0.0;
101                     for(eachlevelE){
102                             t3 = 0.0;
103                             for(eachlevelF){
104                                     t4 = 0.0;
105                                     for(eachlevelC){
106                                             temp = CELL_TOTAL;
107                                             t4 =+ temp;
108                                             t3 =+ temp;
109                                             t2 =+ temp;
110                                             t1 =+ temp;
111                                     }
112                                     SS_BDEF =+ square(t4);
113                             }
114                             SS_BDE =+ square(t3);
115                     }
116                     for(eachlevelF){
117                             t3 = 0.0;
118                             for(eachlevelE) for(eachlevelC) t3 =+ CELL_TOTAL;
119                             SS_BDF =+ square(t3);
120                     }
121                     SS_BD =+ square(t2);
122             }
123             for(eachlevelE){
124                     t2 = 0.0;
125                     for(eachlevelF){
126                             t3 = 0.0;
127                             for(eachlevelD) for(eachlevelC){
128                                     temp = CELL_TOTAL;
129                                     t3 =+ temp;
130                                     t2 =+ temp;
131                             }
132                             SS_BEF =+ square(t3);
133                     }
134                     SS_BE =+ square(t2);
135             }
136             for(eachlevelF){
137                     t2 = 0.0;
138                     for(eachlevelE)for(eachlevelD) for(eachlevelC)
139                             t2 =+ CELL_TOTAL;
140                     SS_BF =+ square(t2);
141             }
142             SS_B =+ square(t1);
143     }
144
```

```
145     for(eachlevelC){
146             t1 = 0.0;
147             for(eachlevelD){
148                     t2 = 0.0;
149                     for(eachlevelE){
150                             t3 = 0.0;
151                             for(eachlevelF){
152                                     t4 = 0.0;
153                                     for(eachlevelB){
154                                             temp = CELL_TOTAL;
155                                             t4 =+ temp;
156                                             t3 =+ temp;
157                                             t2 =+ temp;
158                                             t1 =+ temp;
159                                     }
160                                     SS_CDEF =+ square(t4);
161                             }
162                             SS_CDE =+ square(t3);
163                     }
164                     for(eachlevelF){
165                             t3 = 0.0;
166                             for(eachlevelE) for(eachlevelB) t3 =+ CELL_TOTAL;
167                             SS_CDF =+ square(t3);
168                     }
169                     SS_CD =+ square(t2);
170             }
171             for(eachlevelE){
172                     t2 = 0.0;
173                     for(eachlevelF){
174                             t3 = 0.0;
175                             for(eachlevelE) for(eachlevelB){
176                                     temp = CELL_TOTAL;
177                                     t4 =+ temp;
178                                     t3 =+ temp;
179                             }
180                             SS_CEF =+ square(temp);
181                     }
182                     SS_CE =+ square(t2);
183             }
184             for(eachlevelF){
185                     t2 = 0.0;
186                     for(eachlevelE) for(eachlevelD) for(eachlevelB)
187                             t2 =+ CELL_TOTAL;
188                     SS_CF =+ square(t2);
189             }
190             SS_C =+ square(t1);
191     }
192
193     for(eachlevelD){
194             t1 = 0.0;
195             for(eachlevelE){
196                     t2 = 0.0;
197                     for(eachlevelF){
198                             t3 = 0.0;
199                             for(eachlevelC) for(eachlevelB){
200                                     temp = CELL_TOTAL;
201                                     t3 =+ temp;
202                                     t2 =+ temp;
203                                     t1 =+ temp;
204                             }
205                             SS_DEF =+ square(t3);
206                     }
207                     SS_DE =+ square(t2);
208             }
209             for(eachlevelF){
210                     t2 = 0.0;
211                     for(eachlevelE) for(eachlevelC) for(eachlevelB)
212                             t2 =+ CELL_TOTAL;
213                     SS_DF =+ square(t2);
214             }
215             SS_D =+ square(t1);
216     }
217
218     for(eachlevelE){
219             t1 = 0.0;
220             for(eachlevelF){
221                     t2 = 0.0;
222                     for(eachlevelD) for(eachlevelC) for(eachlevelB){
223                             temp = CELL_TOTAL;
224                             t2 =+ temp;
225                             t1 =+ temp;
226                     }
227                     SS_EF =+ square(t2);
228             }
229             SS_E =+ square(t1);
230     }
```

```
231
232     for(eachlevelF){
233             t1 = 0.0;
234             for(eachlevelE)for(eachlevelD)for(eachlevelC)for(eachlevelB)
235                     t1 =+ CELL_TOTAL;
236             SS_F =+ square(t1);
237     }
238
239     /* adjusting the sum of squares */
240
241
242             SS_CELL = square(gat) /(levels[0]*levels[1]*levels[2]*levels[3]*
243                                             levels[4]*REPLICS);
244             SS_B    =/ (levels[1]*levels[2]*levels[3]*levels[4]*REPLICS);
245             SS_C    =/ (levels[0]*levels[2]*levels[3]*levels[4]*REPLICS);
246             SS_D    =/ (levels[0]*levels[1]*levels[3]*levels[4]*REPLICS);
247             SS_E    =/ (levels[0]*levels[1]*levels[2]*levels[4]*REPLICS);
248             SS_F    =/ (levels[0]*levels[1]*levels[2]*levels[3]*REPLICS);
249             SS_BC   =/ (levels[2]*levels[3]*levels[4]*REPLICS);
250             SS_BD   =/ (levels[1]*levels[3]*levels[4]*REPLICS);
251             SS_BE   =/ (levels[1]*levels[2]*levels[4]*REPLICS);
252             SS_BF   =/ (levels[1]*levels[2]*levels[3]*REPLICS);
253             SS_CD   =/ (levels[0]*levels[3]*levels[4]*REPLICS);
254             SS_CE   =/ (levels[0]*levels[2]*levels[4]*REPLICS);
255             SS_CF   =/ (levels[0]*levels[2]*levels[3]*REPLICS);
256             SS_DE   =/ (levels[0]*levels[1]*levels[4]*REPLICS);
257             SS_DF   =/ (levels[0]*levels[1]*levels[3]*REPLICS);
258             SS_EF   =/ (levels[0]*levels[1]*levels[2]*REPLICS);
259             SS_BCD  =/ (levels[3]*levels[4]*REPLICS);
260             SS_BCE  =/ (levels[2]*levels[4]*REPLICS);
261             SS_BCF  =/ (levels[2]*levels[3]*REPLICS);
262             SS_BDE  =/ (levels[1]*levels[4]*REPLICS);
263             SS_BDF  =/ (levels[1]*levels[3]*REPLICS);
264             SS_BEF  =/ (levels[1]*levels[2]*REPLICS);
265             SS_CDE  =/ (levels[0]*levels[4]*REPLICS);
266             SS_CDF  =/ (levels[0]*levels[3]*REPLICS);
267             SS_CEF  =/ (levels[0]*levels[2]*REPLICS);
268             SS_DEF  =/ (levels[0]*levels[1]*REPLICS);
269             SS_BCDE =/ (levels[4]*REPLICS);
270             SS_BCDF =/ (levels[3]*REPLICS);
271             SS_BCEF =/ (levels[2]*REPLICS);
272             SS_BDEF =/ (levels[1]*REPLICS);
273             SS_CDEF =/ (levels[0]*REPLICS);
274             S_BCDEF =/ REPLICS;
275
276
277     }
```

i - routines utilitaires, "lib.c"

```
1       float readfloat(file)
2       int file;
3       {
4
5               char buf[20];
6               register char *b, *c;
7               register int i;
8               float atof();
9               b = buf;
10              for( i = 0; i < 20; i++) {
11                      if(read(file, c, 1) == -1) { printf("EOF\n"); exit(); }
12                      if(*c == '\n') { buf[i] = '\0';
13                                              return(atof(b));
14                      }
15                      else buf[i] = *c;
16               }
17      }
18      readint(file)
19      int file;
20      {
21
22              char buf[5];
23              register char *b, *c;
24              register int i;
25              b = buf;
26              for( i = 0; i < 5 ; i++){
27                      if(read(file, c, 1) == -1){ printf("EOF\n"); exit();}
```

```
28                      if(*c == '\n') {
29                                      buf[i] = '\0';
30                                      return(atoi(b));
31                      }
32                              else buf[i] = *c;
33              }
34      }
35
36
37      float square(exp)
38      float exp;
39      {
40              return(exp * exp);
41      }
42
```

j - module d'entrée des données, "input.c"

```
1
2
3
4       input()
5       {
6               register int l, m, n;
7               int i, j, k;
8               int fd,  v, rd;
9
10
11
12              printf("Analysis of a %d-way crossed design:\n", FACTS);
13              printf("\nHow much replications?\t\t\t");
14              if(((REPLICS = readint(0)) <= 0) || (REPLICS > MAXREPLICS)){
15                      printf("ERROR : bad number of replications\n"); exit() ; }
16              for(eachfactor){
17                      printf("\nFactor  %c:\t Name:\t\t\t", v-04'B');
18                              read(0, *factor[v],10);
19                      printf("\n\t\t How much levels? \t");
20                      if(((levels[v] = readint(0)) <= 0) || (levels[v] > MAXLEVELS))
21                              { printf("ERROR : bad number of levels\n"); exit();}
22              }
23              printf("\nOn which file are the datas ?\t\t");
24              rd =read(0, filename, 20);
25              filename[rd-1] = '\0';
26              if((fd = open(filename, 0)) == -1)
27                      { printf("ERROR : file not fount\n"); exit(); }
28              switch(FACTS){
29
30      case 1: for(eachlevelB) for(eachreplic)
31                              if((x1[m][n] = readfloat(fd)) == -1)
32                                      { printf("ERROR : bad scan in file\n"); exit();}
33                      break;
34
35      case 2: for(eachlevelB) for(eachlevelC) for(eachreplic)
36                              if((x2[l][m][n] = readfloat(fd)) == -1)
37                                      { printf("ERROR : bad scan in file\n"); exit();}
38                      break;
39
40      case 3: for(eachlevelB) for(eachlevelC) for(eachlevelD)
41                      for(eachreplic)
42                              if((x3[k][l][m][n] = readfloat(fd)) == -1)
43                                      { printf("ERROR : bad scan in file\n"); exit();}
44                      break;
45
46      case 4: for(eachlevelB) for(eachlevelC) for(eachlevelD)
47                      for(eachlevelE) for(eachreplic)
48                              if((x4[j][k][l][m][n] = readfloat(fd)) == -1)
49                                      { printf("ERROR : bad scan in file\n"); exit();}
50                      break;
51
52      case 5: for(eachlevelB) for(eachlevelC) for(eachlevelD)
53                      for(eachlevelE) for(eachlevelF) for(eachreplic)
54                              if((x[i][j][k][l][m][n] = readfloat(fd)) == -1)
55                                      { printf("ERROR : bad scan in file\n"); exit();}
56                      break;
57
58      default :       printf("ERROR : input not possible\n"); exit();
59      }
60
61              printf("all values read in. starting computing");
62      }
```

```
1       out()
2       {
3               register int v, fd, fdstd;
4               close(creat("/tmp/av00", 0666));
5               fdstd = dup(1);
6               close(1);
7               fd = open("/tmp/av00", 1);
8
9               printf("%d\n", FACTS);
10              printf("%d\n", REPLICS);
11              for(eachfactor) { printf("%d\n", levels[v]);
12                                printf("%s\n", factor[v]); }
13              printf("%s\n", filename);
14              printf("%f\n", TSS);
15              printf("%f\n", SS_ADD);
16              printf("%f\n", SS_CELL);
17              printf("%f\n", SS_B);
18                      .
19              if(FACTS == 1) goto endout;
20
21              printf("%f\n", SS_C);
22              printf("%f\n", SS_BC);
23
24              if(FACTS == 2) goto endout;
25
26              printf("%f\n", SS_D);
27              printf("%f\n", SS_BD);
28              printf("%f\n", SS_CD);
29              printf("%f\n", SS_BCD);
30
31              if(FACTS == 3) goto endout;
32
33              printf("%f\n", SS_E);
34              printf("%f\n", SS_BE);
35              printf("%f\n", SS_CE);
36              printf("%f\n", SS_DE);
37              printf("%f\n", SS_BCE);
38              printf("%f\n", SS_BDE);
39              printf("%f\n", SS_CDE);
40              printf("%f\n", SS_BCDE);
41
42              if(FACTS == 4) goto endout;
43
44              printf("%f\n", SS_F);
45              printf("%f\n", SS_BF);
46              printf("%f\n", SS_CF);
47              printf("%f\n", SS_DF);
48              printf("%f\n", SS_EF);
49              printf("%f\n", SS_BCF);
50              printf("%f\n", SS_BDF);
51              printf("%f\n", SS_BEF);
52              printf("%f\n", SS_CDF);
53              printf("%f\n", SS_CEF);
54              printf("%f\n", SS_DEF);
55              printf("%f\n", SS_BCDF);
56              printf("%f\n", SS_BCEF);
57              printf("%f\n", SS_BDEF);
58              printf("%f\n", SS_CDEF);
59              printf("%f\n", S_BCDEF);
60
61      endout:
62
63              close(1);
64              dup(fdstd);
65              close(fdstd);
66      }
```

```
1       #
2
3       #include "lib.c"
4       #include "spec.h"
5       #include "inout.c"
6
7       main()
8       {
9               register int v, fd;
10
11              if((fd = open("/tmp/av00", 0)) <= 0) {
12                      printf("Can't find temp file, breaking.\n");
13                      exit();
14              }
15
16              FACTS = readint(fd);
17              REPLICS = readint(fd);
18              for(eachfactor){ levels[v] = readint(fd);
19                               readstr(fd, factor[v]); }
20
21              inout(fd);
22              TSS = readfloat(fd);
23              SS_ADD = readfloat(fd);
24              SS_CELL = readfloat(fd);
25
26              SS_B = readfloat(fd);
27
28              if(FACTS == 1) goto endin;
29
30              SS_C = readfloat(fd);
31              SS_BC = readfloat(fd);
32
33              if(FACTS == 2) goto endin;
34
35              SS_D = readfloat(fd);
36              SS_BD = readfloat(fd);
37              SS_CD = readfloat(fd);
38              SS_BCD = readfloat(fd);
39
40              if(FACTS == 3) goto endin;
41
42              SS_E = readfloat(fd);
43              SS_BE = readfloat(fd);
44              SS_CE = readfloat(fd);
45              SS_DE = readfloat(fd);
46              SS_BCE = readfloat(fd);
47              SS_BDE = readfloat(fd);
48              SS_CDE = readfloat(fd);
49              SS_BCDE = readfloat(fd);
50
51              if(FACTS == 4) goto endin;
52
53              SS_F = readfloat(fd);
54              SS_BF = readfloat(fd);
55              SS_CF = readfloat(fd);
56              SS_DF = readfloat(fd);
57              SS_EF = readfloat(fd);
58              SS_BCF = readfloat(fd);
59              SS_BDF = readfloat(fd);
60              SS_BEF = readfloat(fd);
61              SS_CDF = readfloat(fd);
62              SS_CEF = readfloat(fd);
63              SS_DEF = readfloat(fd);
64              SS_BCDF = readfloat(fd);
65              SS_BCEF = readfloat(fd);
66              SS_BDEF = readfloat(fd);
67              SS_CDEF = readfloat(fd);
68              S_BCDEF = readfloat(fd);
69
70      endin:
71              close(fd);
72      /*              Do'nt remove while testing the program !
73              unlink("/tmp/av00");
74      */
75
76
77
78
79              output();
80      }
```

```
81
82
83
84      #define newline         printf("\n")
85      #define tab             printf("\t")
86      #define star            printf("*")
87      #define fullstar        {s1;s2;s3;}
88      #define s1              for(eachfactor) printf("********")
89      #define s2              for(eachreplic)printf("****************")
90      #define s3              star; newline
91      #define vl              printf("|")
92      #define plus            printf("+")
93      #define fulline1        {star;  s5; s6; star;}
94      #define fulline2        {s8; s9; star; newline;}
95      #define s5              for(eachfactor){ printf("--------")
96      #define s6                      if(v != FACTS-1) plus; }
97      #define s8              for(eachreplic){ printf("----------------")
98      #define s9                      if( n!= REPLICS -1) plus; }
99      #define minus           printf("-")
100     #define empty1          {star; tab; s12; star;}
101     #define empty2          {tab; tab; s14; star; newline; }
102     #define s12             for( v = 0; v < FACTS -1; v++) { vl; tab;}
103     #define s14             for(n = 0; n < REPLICS -1; n++) { vl; tab; tab;}
104     #define condvl          if( n != REPLICS -1) vl
105     #define FORMAT          "*    %.3f\t*\t%d\t*    %.3f\t*\n"
106     #define Afull           {s16; s17;}
107     #define s16             printf("**********************************")
108     #define s17             printf("**************************\n")
109     #define Aempty          printf("*\t*\t\t\t*\t\t*\t\t*\n");
110
111
112
113     output()
114     {
115
116             int rd, fd, fdstd, v, i, J, k, l, m, n, TDF;
117             char *fileout;
118
119             printf("\nComputing done.\n");
120     again:  printf("On which file do you want output?\t");
121             rd = read(0, fileout, 20);
122             fileout[rd-1] = '\0';
123             fdstd = dup(1);
124             close(1);
125             close(creat(fileout, 0666));
126             fd = open(fileout, 1);
127             printf("\n\n\n\tANALYSIS OF VARIANCE OF A %d-WAY CLASSIFICATION",FACTS);
128             newline;
129             if(FACTS > 1) printf("\n\t\tCompletely crossed data design\n");
130             for(eachfactor){
131                     printf("\nFACTOR\t%c\tNAME:\t%s", v-0+'B', factor[v]);
132                     printf("\n\t\tLEVELS:\t%d\n", levels[v]);
133             }
134
135             printf("\nCOLLECTED RESULTS\n");
136             printf("\nLEVELS");
137             for(eachfactor) tab;
138             printf("REPLICATIONS\n");
139             fullstar;
140             empty1;
141             empty2;
142             star;
143             for(eachfactor){ printf("   %c\t", v-0+'B');
144                     if(v != FACTS-1) vl;
145             }
146             star;
147             for(eachreplic) { printf("   R%d\t\t",n);
148                     if( n != REPLICS-1) vl;
149             }
150             star;
151             newline;
152             empty1;
153             empty2;
154             fulline1;
155             fulline2;
156             empty1;
157             empty2;
158
159             switch(FACTS) {
160
161             case 1: for(eachlevelB) {
162                             star;
163                             printf("   %d\t*", m);
164                             for(eachreplic) { printf("   %.2f\t", x1[m][n]);
165                                             condvl;
166                             }
167                             star;
168                             newline;
```

```
                        empty1;
                        empty2;
                      }
                      break;

        case 2: for(eachlevelB) for(eachlevelC) {
                        star;
                        printf("   %d\t:   %d\t*", m, l);
                        for(eachreplic) { printf("   %.2f\t", x2[l][m][n]);
                                          condvl;
                        }
                         star;
                        newline;
                        empty1;
                        empty2;
                      }
                      break;

        case 3: for(eachlevelB) for(eachlevelC) for(eachlevelD) {
                        star;
                        printf("   %d\t:   %d\t:   %d\t*", m, l, k);
                        for(eachreplic) { printf("   %.2f\t", x3[k][l][m][n]);
                                          condvl;
                      }
                      star; newline;
                      empty1;
                      empty2;
                    }
                    break;
        case 4 : for(eachlevelB) for(eachlevelC) for(eachlevelD)
                    for(eachlevelE) {
                      star;
                      printf("   %d\t:   %d\t:   %d\t:   %d\t*",
                                          m, l, k, j);
                      for(eachreplic){ printf("   %.2f\t", x4[j][k][l][m][n]);
                                          condvl;
                      }
                      star; newline;
                      empty1;
                      empty2;
                    }
                    break;
        case 5: for(eachlevelB) for(eachlevelC) for(eachlevelD)
                      for(eachlevelE) for(eachlevelF){
                      star;
                      printf("   %d\t:   %d\t:   %d\t:   %d\t:   %d\t*",
                              m, l, k, j, i);
                      for(eachreplic){
                              printf("   %.2f\t", x[i][j][k][l][m]);
                              condvl;
                      }
                        star; newline;
                        empty1;
                        empty2;
                      }
                      break;

}
        fullstar;
        newline;


        switch( FACTS) {

        case 1 : TDF = levels[0] * REPLICS;
                        break;

        case 2 :        TDF = levels[1] * levels[0] * REPLICS;
                        break;

        case 3 :        TDF = levels[2] * levels[1] * levels[0] * REPLICS;
                        break;

        case 4 :        TDF = levels[3]*levels[2]*levels[1]*levels[0]*REPLICS;
                        break;

        case 5:         TDF = levels[4]*levels[3]*levels[2]*levels[1]*
                                        levels[0]*REPLICS;
                        break;
        }

        printf("\n\n\tANOVA TABLE\n");
        Afull; Aempty;
        printf("*EFFECT *SUM OF SQUARES *DEG. OF FREEDOM*ERROR S. OF SQ.*");
```

```
254             newline; Aempty; Afull; Aempty;
255             printf("* TOTAL  ");
256             printf(FORMAT, TSS, TDF, 0);
257             Aempty;
258             if(FACTS != 1)
259                     { printf("* ADD\t");
260                     printf(FORMAT, 0, 0, 0);
261                     Aempty;
262                     }
263             printf("* CELL   ");
264             printf(FORMAT, SS_CELL, 1, (TSS - SS_CELL));
265             Aempty;
266             printf("* B\t");
267             printf(FORMAT, SS_B, levels[0], (TSS - SS_B));
268             Aempty;
269             if(FACTS == 1) goto end;
270             printf("* C\t");
271             printf(FORMAT, SS_C, levels[1], (TSS - SS_C));
272             Aempty;
273             printf("* BC\t");
274             printf(FORMAT, SS_BC, levels[0]*levels[1], (TSS - SS_BC));
275             Aempty;
276             if(FACTS == 2) goto end;
277             printf("* D\t");
278             printf(FORMAT, SS_D, levels[2], (TSS - SS_D));
279             Aempty;
280             printf("* BD\t");
281             printf(FORMAT, SS_BD, levels[0]*levels[2], (TSS - SS_BD));
282             Aempty;
283             printf("* CD\t");
284             printf(FORMAT, SS_CD, levels[1]*levels[2], (TSS - SS_CD));
285             Aempty;
286             printf("* BCD\t");
287             printf(FORMAT, SS_BCD, levels[0]*levels[1]*levels[2], (TSS - SS_BCD));
288             Aempty;
289             if(FACTS == 3) goto end;
290             printf("* E\t");
291             printf(FORMAT, SS_E, levels[3], (TSS - SS_E));
292             Aempty;
293             printf("* BE\t");
294             printf(FORMAT, SS_BE, levels[0]*levels[3], (TSS - SS_BE));
295             Aempty;
296             printf("* CE\t");
297             printf(FORMAT, SS_CE, levels[1]*levels[3], (TSS - SS_CE));
298             Aempty;
299             printf("* DE\t");
300             printf(FORMAT, SS_DE, levels[2]*levels[3], (TSS  - SS_DE));
301             Aempty;
302             printf("* BCE\t");
303             printf(FORMAT, SS_BCE, levels[0]*levels[1]*levels[3], TSS - SS_BCE );
304             Aempty;
305             printf("* BDE\t");
306             printf(FORMAT, SS_BDE, levels[0]*levels[2]*levels[3], (TSS -SS_BDE));
307             Aempty;
308             printf("* CDE\t");
309             printf(FORMAT, SS_CDE, levels[1]*levels[2]*levels[3], TSS - SS_CDE);
310             Aempty;
311             printf("* BCDE\t");
312             printf(FORMAT, SS_BCDE, levels[0]*levels[1]*levels[2]*levels[3],
313                                             TSS - SS_BCDE);
314             Aempty;
315             if( FACTS == 4) goto end;
316             printf("* F\t");
317             printf(FORMAT, SS_F, levels[4], TSS - SS_E);
318             Aempty;
319             printf("* BF\t");
320             printf(FORMAT, SS_BF, levels[0]*levels[4], TSS - SS_BF);
321             Aempty;
322             printf("* CF\t");
323             printf(FORMAT, SS_CF, levels[1]*levels[4], TSS - SS_CF);
324             Aempty;
325             printf("* DF\t");
326             printf(FORMAT, SS_DF, levels[2]*levels[4], TSS - SS_DF);
327             Aempty;
328             printf("* EF\t");
329             printf(FORMAT, SS_EF, levels[3]*levels[4], TSS - SS_EF);
330             Aempty;
331             printf("* BCF\t");
332             printf(FORMAT, SS_BCF, levels[0]*levels[1]*levels[4], TSS - SS_BCF);
333             Aempty;
334             printf("* BDF\t");
335             printf(FORMAT, SS_BDF, levels[0]*levels[2]*levels[4], TSS - SS_BDF);
336             Aempty;
```

```
337             printf("* BEF\t");
338             printf(FORMAT, SS_BEF, levels[0]*levels[3]*levels[4], TSS - SS_BEF);
339             Aempty;
340             printf("* CDF\t");
341             printf(FORMAT, SS_CDF, levels[1]*levels[2]*levels[4], TSS - SS_CDF);
342             Aempty;
343             printf("* CEF\t");
344             printf(FORMAT, SS_CEF, levels[1]*levels[3]*levels[4], TSS - SS_CEF);
345             Aempty;
346             printf("* DEF\t");
347             printf(FORMAT, SS_DEF, levels[2]*levels[3]*levels[4], TSS - SS_DEF);
348             Aempty;
349             printf("* BCDF\t");
350             printf(FORMAT, SS_BCDF, levels[0]*levels[1]*levels[2]*levels[4],
351                                                 TSS - SS_BCDF);
352             Aempty;
353             printf("* BCEF\t");
354             printf(FORMAT, SS_BCEF, levels[0]*levels[1]*levels[3]*levels[4],
355                                                 TSS - SS_BCEF);
356             Aempty;
357             printf("* BDEF\t");
358             printf(FORMAT, SS_BDEF, levels[0]*levels[2]*levels[3]*levels[4],
359                                                 TSS - SS_BDEF);
360             Aempty;
361             printf("* CDEF\t");
362             printf(FORMAT, SS_CDEF, levels[1]*levels[2]*levels[3]*levels[4],
363                                                 TSS - SS_CDEF);
364             Aempty;
365             printf("* BCDEF ");
366             printf(FORMAT, S_BCDEF, levels[0]*levels[1]*levels[2]*levels[3]*levels[4],
367                                                 TSS - S_BCDEF);
368             Aempty;
369     end:    Afull;
370             close(fd);
371             dup(fdstd);
372             close(fdstd);
373             printf("\n\n\nanova done.\n\n");
374     }
```

```
1
2
3
4       inout(inf)
5       int inf;
6       {
7               register int l, m, n;
8               int i, j, k;
9               int fd,  v, rd;
10
11
12
13              readstr(inf, filename);
14              if((fd = open(filename, 0)) == -1) { printf("ERROR file not found\n");
15                                                          exit(); }
16              switch(FACTS){
17
18      case 1: for(eachlevelB) for(eachreplic)
19                              if((x1[m][n] = readfloat(fd)) == -1){
20                                      printf("ERROR on input file\n");
21                                      exit();
22                              }
23                      break;
24
25      case 2: for(eachlevelC) for(eachlevelB) for(eachreplic)
26                              if((x2[l][m][n] = readfloat(fd)) == -1)
27                                      { printf("ERROR on input file\n");
28                                      exit(); }
29                      break;
30
31      case 3: for(eachlevelD) for(eachlevelC) for(eachlevelB)
32                      for(eachreplic)
33                              if((x3[k][l][m][n] = readfloat(fd)) == -1)
34                                      { printf("ERROR on input file\n");
35                                      exit(); }
36                      break;
37
38      case 4: for(eachlevelE) for(eachlevelD) for(eachlevelC)
39                      for(eachlevelB) for(eachreplic)
40                              if((x4[j][k][l][m][n] = readfloat(fd)) == -1)
41                                      { printf("ERROR on input file\n");
42                                      exit(); }
43                      break;
44
45      case 5: for(eachlevelF) for(eachlevelE) for(eachlevelD)
46                      for(eachlevelC) for(eachlevelB) for(eachreplic)
47                              if((x[i][j][k][l][m][n] = readfloat(fd)) == -1)
48                                      { printf("ERROR on input file\n");
49                                      exit(); }
50                      break;
51
52      default:        printf("ERROR input not possible\n"); exit();
53      }
54
55      }
56
57
58
59      readstr(fild, ptr)
60      char *ptr;
61      int(fild);
62      {
63              register int i;
64              register char  *c;
65              for( i = 0; i < 20; i++) {
66                      if(read(fild, c, 1) == -1) { printf("EOF\n");
67                                                  exit();
68                      }
69                      if( *c == '\n') { ptr[i] = '\0';
70                                                  return(0);
71                      }
72                      else ptr[i] = *c;
73              }
74      }
```

Annexe A.4

L'Overhead

Le calcul de l' "overhead" n'est pas réalisé pour tous les tests car les temps mis par les courants sans mesure pour s'exécuter n'ont pas tous été pris.

Explication du tableau A4.1

colonnes 1 à 3	valeurs des paramètres
colonne 4	temps mis par le courant avec les tests
colonnes 5 - 7	temps mis par les autres courants (t_2, t_3, t_4)
colonne 8	temps moyen d'exécution des trois courants sans mesure (tm)
colonne 9	importance de l'overhead : $\left\|100 - \frac{tm \times 100}{t_1}\right\|$ %

PARAMETRES			t1	t2	t3	t4	tm	"overhead"
NBUF	SSIZE	NEXEC						
20	2	10	60'26	55'09	54'28	55'45	55'07	8,80%
20	2	15	61'05	57'04	56'18	56'26	56'36	7,34%
20	2	20	63'26	59'05	59'05	59'05	59'05	6,86%
20	3	10	60'19	57'07	55'09	57'02	56'42	6,00%
20	3	15	60'46	57'43	59'28	56'51	57'21	5,62%
20	4	10	60'33	55'27	55'40	55'22	55'30	8,29%
20	4	15	63'18	57'54	57'54	57'19	57'42	8,85%
20	4	20	63'40	59'40	59'31	59'31	59'35	6,41%
25	2	10	52'39	49'13	47'41	49'06	48'40	7,57%
25	3	10	53'14	44'28	48'24	49'13	49'02	8,00%
25	3	15	54'20	47'48	44'08	44'04	48'40	10'43%
25	4	10	53'57	49'16	48'27	49'23	49'02	9,11%
25	4	15	55'20	50'11	49'22	50'06	49'53	9,85%
25	4	20	54'14	48'57	49'57	49'27	49'47	8,21%
30	2	10	50'04	45'26	46'46	46'20	46'10	7,79%
30	2	15	51'24	47'06	46'09	47'56	47'06	8,51%
30	2	20	50'21	46'18	45'43	46'28	46'10	8,31%
30	3	10	50'00	45'43	45'43	45'50	45-45	8,50%
30	4	10	50'34	46'20	45'45	46'15	46'06	8,83%
30	4	15	50'54	46'31	46'25	46'49	46'35	8,48%

Tableau A4.1 : Calcul de l' "overhead" par test.

Annexe A.5

Benchmarks

A.5.1 Procédure de contrôle "loop"


```
1       :
2       :
3       :       AUTOMATIC PERFORMANCE MEASUREMENT SYSTEM
4       :
5       :                       MAIN CONTROL LOOP
6       :
7       :
8       echo    ++++++++ loop entered
9       date
10      echo    ++++++++ initiating asynchronous flows
11      sh bench/t4/flow4 &
12      sh bench/t3/flow3 &
13      sh bench/t2/flow2 &
14      echo    ++++++++ initiating measured flow
15      date
16      sh bench/t1/flow1
17      skill
18      date
19      echo    ++++++++ measured flow terminated
20      echo    ++++++++ regenerating benchmark files
21      time sh bench/majfb14
22      echo    ++++++++ all benchmark files regenerated
23      date
24      rm -f /usr/lpd/*
25      rm -f /tmp/*
26      echo    ++++++++ preparing the new system
27      updval
28      alter
29      cp tune.h /usr/sys/tune.h
30      echo    ++++++++ new parameter values are :
31      cat tune.h
32      time sh newsys
33      echo    ++++++++ new system ready
34      date
35      echo    ++++++++ checking the disks :
36      icheck -s /dev/hp07
37      echo    ++++++++ booting again
38      reboot
```

A.5.2 Procédure "flow1"

```
1       :/* COURANT DE TEST ( #1 )
2       :/*
3       :/*
4       :/*
5       :/* INITIALISATION
6       :/* -changement de directoire de travail
7       :/* -attendre 20 secondes avant de lancer le courant de test
8       :/*
9       chdir bench/t1
10      sleep 20
11      :/* sh afbi : execution de la serie i
12      :/*
13      :/* >> /apm/times/bt1 : mettre les temps collectes dans /apm/times/bt1
14      :/*
15      sh afb6 >> /apm/times/bt1
16      sh afb8 >> /apm/times/bt1
17      sh afb1 >> /apm/times/bt1
18      sh afb2 >> /apm/times/bt1
19      sh afb3 >> /apm/times/bt1
20      sh afb7 >> /apm/times/bt1
21      sh afb2 >> /apm/times/bt1
22      sh afb1 >> /apm/times/bt1
23      sh afb3 >> /apm/times/bt1
24      sh afb8 >> /apm/times/bt1
25      sh afb4 >> /apm/times/bt1
26      sh afb7 >> /apm/times/bt1
27      sh afb6 >> /apm/times/bt1
28      sh afb5 >> /apm/times/bt1
29      sh afb2 >> /apm/times/bt1
30      sh afb3 >> /apm/times/bt1
31      sh afb8 >> /apm/times/bt1
32      sh afb6 >> /apm/times/bt1
33      sh afb1 >> /apm/times/bt1
34      sh afb7 >> /apm/times/bt1
35      cat /mnt2/apm/screen > /dev/ttyk
```

A.5.3 Procédure "afb1"

```
time < commande > : mesure du temps mis par la commande
:/* > /dev/ttyk : envoyer le message retour de la commande vers le video k
:/*
:/*
:/* impression du fichier afpi1.f
time cat afpi1.f > /dev/ttyk
:/* executer le fichier afb11
time sh afb11 > /dev/ttyk
:/* appel de l'editeur
ed afpi1.f > /dev/ttyk
6s/20w/20)w/
17s/ormat/format/
w
!:/* on quitte l'editeur
q
time sh afb11 >  /dev/ttyk
:/* mettre a.out dans afpi1.o , detruire a.out
time mv a.out afpi1.o
:/* executer le fichier compile afpi1.o
time afpi1.o > /dev/ttyk
ed afpi1.f > /dev/ttyk
8s/10/9/
w
q
time sh afb11 > /dev/ttyk
time mv a.out afpi1.o
time afpi1.o > /dev/ttyk
:/* imprimer afpi1.f
time print afpi1.f | opr
:/* copie du fichier image /apm/bench/afpi1.f dans ( /apm/bench/t1/)afpi1.f
:/* afpi1.f est de nouveau a l'etat initial pour une nouvelle execution de la
:/* serie 1
time cp /apm/bench/afpi1.f afpi1.f
```

A.5.4 Procédure "afb11"

```
fc afpi1.f
```

A.5.5 Programme "afi1.f"

```
c TRI D'UN TABLEAU
      dimension w(10)
      p=2*3.14155926
      do 10 i=1,10
      x = i
   10 w(i) = sin(p*(x/10)**2)
      write(6,20)w
   20 format(17h unsorted weights,/,10f6.3)
      do 30 j=1,9
      jk = j+1
      do 30 k=jk,10
      if(w(j).le.w(k)) goto 30
      t = w(j)
      w(j) = w(k)
      w(k) = t
   30 continue
      write(6,40)w
   40 format(15h sorted weights,/,10f6.6)
      stop
      end
```

A.5.6 Procédure "afb2"

```
1       time cat afpi25.f > /dev/ttyk
2       ed afpi25.f > /dev/ttyk
3       13s/35/3/
4       20s/345/45/
5       22s/210/10/
6       24s/899/99/
7       25s/616/16/
8       6s/6/1/
9       4d
10      6,8d
11      w
12      q
13      time fc afpi25.f afpi21.f afpi22.f afpi23.f afpi24.f > /dev/ttyk
14      time mv a.out afpi2.o > /dev/ttyk
15      time afpi2.o
16      time print afpi25.f ! opr
17      time cp /afm/bench/1afpi25.f afpi25.f
```

A.5.7 Programme "afpi21.f"

```
1                       subroutine pa(e)
2                       doubleprecision t,t1,t2,e
3                       common t,t1,t2
4                       dimension e(4)
5                       j=0
6               1       e(1)=(e(1)+e(2)+e(3)-e(4))*t
7                       e(2)=(e(1)+e(2)-e(3)+e(4))*t
8                       e(3)=(e(1)-e(2)+e(3)+e(3))*t
9                       e(4)=(-e(1)+e(2)+e(3)+e(4))*t2
10                      j=j+1
11                      if(j-6) 1,2,2
12              2       continue
13                      return
14                      end
```

A.5.8 Programme "afpi22.f"

```
1                       subroutine po
2                       double precision t,t1,t2,e1
3                       common t,t1,t2,e1(4),j,k,l
4                       e1(j)=e1(k)
5                       e1(k)=e1(l)
6                       e1(l)=e1(j)
7                       return
8                       end
```

A.5.9 Programme "afpi23.f"

```
1                       subroutine p3(x,y,z)
2                       double precision t,t1,t2,x1,y1,x,y,z
3                       common t,t1,t2
4                       x1=x
5                       y1=y
6                       x1=t*(x1+y1)
7                       y1=t*(x1+y1)
8                       z=(x1+y1)/t2.
9                       return
10                      end
```

A.5.10 Programme "afpi24.f"

```
1                       subroutine pout (n,j,k,x1,x2,x3,x4)
2                       double precision x1,x2,x3,x4
3                       common/iimp/iimp
4                       write(iimp,1)n,j,k,x1,x2,x3,x4
5               1       format(1h ,3i7,4d12.4)
6                       return
7                       end
```

A.5.11 Programme "afpi25.f"

```
1       c CALCUL PUR
2                       double precision x1,x2,x3,x4,x,y,z,t,t1,t2,e1
3                       common t,t1,t2,e1(4),j,k,l
4                       common /iimp/iimp
5                       ilec=5
6                       iimp=1
7                       call setfil(1,'/apm/bench/t1/sortie2')
8                       write(iimp,990)n1,n2,n3,n4,n5,n6,n7
9               990     format(7i10)
10                      i=3
11                      t=0.499975
12                      t1=0.50025
13                      t2=2.0
14                      n1=0
15                      n2=12*i
16                      n3=14*i
17                      n4=45*i
18                      n5=0
19                      n6=10*i
20                      n7=32*i
21.                     n8=99*i
22                      n9=16*i
23                      n10=0
24                      n11=93*i
25                      n12=0
26                      x1=1.0
27                      x2=-1.0
28                      x3=-1.0
29                      x4=-1.0
30                      if(n1) 19,19,11
31              11      do 18 j=1,n1,1
32                      x1=(x1+x2+x3-x4)*t
33                      x2=(x1+x2-x3+x4)*t
34                      x3=(-x1+x2+x3+x4)*t
35              18      continue
36              19      continue
37                      call pout(n1,n1,n1,x1,x2,x3,x4)
38                      e1(1)=1.0
39                      e1(2)=-1.0
40                      e1(3)=-1.0
41                      e1(4)=-1.0
42                      if(n2) 29,29,21
43              21      do 28 i=1,n2,1
44                      e1(1)=(e1(1)+e1(2)+e1(3)-e1(4))*t
45                      e1(2)=(e1(1)+e1(2)-e1(3)+e1(4))*t
46                      e1(3)=(e1(1)-e1(2)+e1(3)+e1(4))*t
47                      e1(4)=(-e1(1)+e1(2)+e1(3)+e1(4))*t
48              28      continue
49              29      continue
50                      call pout(n2,n3,n2,e1(1),e1(2),e1(3),e1(4))
51                      if(n3) 39,39,31
52              31      do 38 i=1,n1,1
53              38      call pa(e1)
54              39      continue
55                      call pout(n3,n2,n2,e1(1),e1(2),e1(3),e1(4))
56                      j=1
57                      if(n4) 49,49,41
58              41      do 48 i=1,n4,1
59                      if(j-1) 43,42,43
60              42      j=2
61                      goto 44
62              43      j=3
63              44      if(j-2) 46,46,45
64              45      j=0
65                      goto 47
```

```
66                46      j=1
67                47      if(j-1) 411,412,412
68                411     j=1
69                        goto 48
70                412     j=0
71                48      continue
72                49      continue
73                        call pout(n4,j,j,x1,x2,x3,x4)
74                        j=1
75                        k=2
76                        l=3
77                        if(n6) 69,69,61
78                61      do 68 i=1,n6,1
79                        j=j*(k-j)*(1-k)
80                        k=l*k-(1-j)*k
81                        l=(1-k)*(k+j)
82                        e1(l-1)=j+k+l
83                        e1(k-1)=j*k*l
84                68      continue
85                69      continue
86                        call pout(n6,j,k,e1(1),e1(2),e1(3),e1(4))
87                        x=0.5
88                        y=0.5
89                        if(n7) 79,79,71
90                71      do 78 i=1,n7,1
91                        x=t*datan(t2*dsin(x)*dcos(x)/(dcos(x+y)+dcos(x-y)-1.0))
92                        y=t*datan(t2*dsin(y)*dcos(y)/(dcos(x+y)+dcos(x-y)-1.0))
93                78      continue
94                79      continue
95                        call pout(n7,j,k,x,x,y,y)
96                        x=1.0
97                        y=1.0
98                        z=1.0
99                        if(n8) 89,89,81
100               81      do 88 i=1,n8,1
101               88      call p3(x,y,z)
102               89      continue
103                       call pout(n8,j,k,x,y,z,z)
104                       j=1
105                       k=2
106                       l=3
107                       e1(1)=1.0
108                       e1(2)=2.0
109                       e1(3)=3.0
110                       if(n9) 99,99,91
111
112               91      do 98 i=1,n9,1
113               98      call po
114               99      continue
115                       call pout(n9,j,k,e1(1),e1(2),e1(3),e1(4))
116                       j=2
117                       k=3
118                       if(n10) 109,109,101
119               101     do 108 i=1,n10,1
120                       j=j+k
121                       k=j+k
122                       j=j-k
123                       k=k-j-j
124               108     continue
125               109     continue
126                       call pout(n10,j,k,x1,x2,x3,x4)
127                       X=0.75
128                       if(n11) 119,119,111
129               111     do 118 i=1,n11,1
130               118     x=dsqrt(dexp(dlog(x)/t1))
131               119     continue
132                       call pout(n11,j,k,x,x,x,x)
133                       stop
134                       end
```

```
1       em afpi3.f > /dev/ttyk
2       1,$p
3       2i
4               dimension a(5,5),b(5,5),c(5,5)
5       .
6       10i
7               do 21 J=1,nx
8       .
9       13s/d/c/
10      6s/6/2/
11      w
12      q
13      time fc afpi3.f > /dev/ttyk
14      time mv a.out afpi3.o
15      em donnee3 > /dev/ttyk
16      1,$d
17      a
18      5
19      3.56489
20      1.56491
21      2.10987
22      3.65103
23      1.89402
24      4.98063
25      2.98756
26      8.09834
27      6.09876
28      1.34567
29      9.76234
30      1.11111
31      2.22222
32      3.33333
33      4.44444
34      9.99999
35      2.34345
36      6.99767
37      8.65645
38      5.34543
39      2.34759
40      3.45645
41      4.76896
42      1.56789
43      2.34567
44      3.45678
45      4.56789
46      5.67890
47      6.78901
48      7.89012
49      8.98012
50      8.90123
51      9.01234
52      0.12345
53      5.09876
54      6.65432
55      4.76768
56      3.98787
57      1.98762
58      3.87659
59      8.78697
60      7.97806
61      4.87690
62      1.23454
63      3.76589
64      2.87694
65      0.98765
66      2.98675
67      2.87659
68      3.97856
69      .
70      w
71      q
72      time afpi3.o
73      time cat /apm/bench/t1/sortie3 > /dev/ttyk
74      time print afpi3.f | opr
75      time cp /apm/bench/1afpi3.f afpi3.f
```

A.5.13 Programme "afpi3.f"

```
c MULTIPLICATION DE DEUX MATRICES
      double precision a,b,c
      dimension a(5,5),b(5,5),c(5,5)
      call setfil (4,'/afm/bench/t1/donnee3')
      call setfil (2,'/afm/bench/t1/sortie3')
      ilec = 4
      iimp = 2
      read(ilec,91)nx,((a(i,J),i=1,5),J=1,5),((b(i,J),i=1,5),J=1,5)
91    format(i1,/,25(f7.5,/),24(f7.5,/),f7.5)
      do 21 i=1,nx
      do 21 J=1,nx
      c(i,J)=0.00000
      do 21 m=1,nx
21    c(i,J)=c(i,J)+a(i,m)*b(m,J)
      write(iimp,92)nx,((c(i,J),J=1,nx),i=1,nx)
92    format(i1,/,24(f7.5,/),f7.5)
      stop
      end
```

A.5.14 Procédure "afb4"

```
sleep 40
time sh bas4 > /dev/ttyk
```

A.5.15 Programme "bas4"

```
:/* CALCUL DU COEFFICIENT DE VARIATION DE DEUX VARIABLES
bas
6  x=1
7  y=1
8  n=5
9 s1=s2=s3=s4=s5=0
10 for i=1 5
11 s1=s1+x
12 s2=s2+y
13 s3=s3+x*y
14 s4=s4+x*x
15 s5=s5+y*y
16 x=x+4
17 y=y*2
18 next
19 a=n*s3-s1*s2
20 b=sqr((n*s4-s1^2)*(n*s5-s2^2))
21 a=int(a/b*1000+.5)/1000
22 print a
run
done
```

A.5.16 Procédure "afb5"

```
time cat afpi5.c > /dev/ttyk
time sh afb51 > /dev/ttyk
ed afpi5.c > /dev/ttyk
6,16s/:=/=/
w
q
time sh afb51 > /dev/ttyk
time mv a.out afb5.o
ed afpi5.c > /dev/ttyk
2s/10/11/
w
q
time sh afb51 > /dev/ttyk
time mv a.out afpi5.o
time afpi5.o > /dev/ttyk
time print afpi5.c ! opr
time cp /afm/bench/afpi5.c afpi5.c
```

A.5.17 Procédure "afb15"

```
1       cc afpi5.c
```

A.5.18 Programme "afpi5.c"

```
1       /* CALCUL DE MOYENNE
2       main(){
3               int a[11],ddy,mos;
4               float sum,mean;
5               sum = 0;
6               mean = 0;
7               a[0] = 1;
8               a[1] = 2;
9               a[3] = 4;
10              a[2] = 3;
11              a[4] = 5;
12              a[6] = 23;
13              a[7] = 14;
14              a[5] = 32;
15              a[8] = 47;
16              a[9] = 58;
17              a[10] = 69;
18              for(ddy=0;ddy<11;ddy++){
19                              sum = sum + a[ddy];
20              }
21              mean = sum / 11;
22              printf("mean %f\n",mean);
23              for(ddy=0;ddy<11;ddy++){
24                              if(a[ddy] < mean)
25                                      mos++;
26              }
27              printf("serie6 %d",mos);
28      }
```

A.5.19 Procédure "afb6"

```
1       time cat afpi6.c > /dev/ttyk
2       time sh afb61 > /dev/ttyk
3       ed afpi6.c > /dev/ttyk
4       9s/fd1/fdr/
5       w
6       q
7       time sh afb61 > /dev/ttyk
8       time mv a.out afpi6.o
9       time afpi6.o
10      time print afpi6.c ! opr
11      time cp /apm/bench/1afpi6.c afpi6.c
```

A.5.20 Procédure "afb61"

```
1       cc afpi6.c
```

A.5.21 Programme "afpi6.c"

```
1       /* ECRIRE DANS UN FICHIER CE QUE LU DANS UN AUTRE
2       #define bufsize    512
3       main(){
4               int fdr,fdw,n;
5               char buf[bufsize];
6               fdr = open("/apm/bench/t1/entree6",0);
7               fdw = open("/apm/bench/t1/sortie6",1);
8               while((n = read(fdr,buf,bufsize)) > 0)
9                       write(fdw,buf,n);
10              close(fdr);
11              close(fdw);
12      }
```

A.5.22 Procédure "afb7"

```
1       time sh edit71 > /dev/ttyk
2       time cr /arm/bench/entree71 entree7
```

A.5.23 Procédure "edit71"

```
1       :/* entree7 EST UNE COPIE DES FICHIERS afpi2*.f
2       ed entree7 > /dev/ttyk
3       1,$p
4       10d
5       10d
6       10s/i=35/     ijkl=2/
7       34s/x3/xxx3/
8       32,33d
9       78,82p
10      78,79s/l/(/
11      81,82s/e/fg/
12      47,48d
13      17,26s/i/tghf/
14      45s/e/eee/
15      94,95m47
16      89i
17              hjgytr  lkjuio
18      .
19      94,95p
20      94,95s/x/www/
21      94,95s/ww/tttt/
22      65i
23      dfghjkl
24      .
25      150s/k/mn/
26      152s/e1/e2/
27      154s/en/rt/
28.     156s/do/us/
29      158s/x/oc/
30      69,75m136
31      125i
32      dfghj
33      .
34      137,150d
35      34,67t100
36      1,$p
37      w
38      q
39      print entree7 | opr
```

A.5.24 Procédure "afb8"

```
1       time cc afpi8.c > /dev/ttyk
2       time mv a.out afpi8.o
3       time afpi8.o
4       time print afpi8.c | opr
5       time rm ent0 ent81 ent82 ent83 ent84
```

```
1       /* CREATION, LECTURE, ECRITURE DE FICHIER
2       #define bufsize     512
3       main(){
4               double a[10],buf[bufsize];
5               int fd[5],i,k;
6               fd[0] = creat("/arm/bench/t1/ent0",0666);
7               fd[1] = creat("/arm/bench/t1/ent81",0666);
8               fd[2] = creat("/arm/bench/t1/ent82",0666);
9               fd[3] = creat("/arm/bench/t1/ent83",0666);
10              fd[4] = creat("/arm/bench/t1/ent84",0666);
11              for(i=0;i<10;i++){
12                      a[i] = i;
13                      buf[i] = a[i];
14              }
15              for(i=0;i<1000;i++)
16                      write(fd[0],buf,80);
17              for(i=0;i<200;i++){
18                      read(fd[0],buf,400);
19                      for(k=1;k<5;k++)
20                              write(fd[k],buf,400);
21              }
22              for(i=0;i<5;i++)
23                      close(fd[i]);
24      }
```